Calhoun
Institutional Archive of the Naval Postgraduate School

Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations 1. Thesis and Dissertation Collection, all items

1973-03

A natural language system for experimenting with semantic representations based on the REL English syntax and a general purpose parser in LISP

Courts, David Paul

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/16717

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun

DUDLEY KNOX LIBRARY

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

A NATURAL LANGUAGE SYSTEM FOR EXPERIMENTING WITH SEMANTIC REPRESENTATIONS BASED ON THE REL ENGLISH SYNTAX AND A GENERAL PURPOSE PARSER IN LISP

David Paul Courts

Library
Naval Postgraduate School
Monterey, California 93940

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A NATURAL LANGUAGE SYSTEM FOR EXPERIMENTING WITH SEMANTIC REPRESENTATIONS BASED ON THE REL ENGLISH SYNTAX AND A GENERAL PURPOSE PARSER IN LISP

by

David Paul Courts

Thesis Advisor: G.D. Gibbons

March 1973

Approved for public release; distribution unlimited.

T153373

A Natural Language System
for
Experimenting with Semantic Representations
based on the
REL English Syntax
and a
General Purpose Parser in LISP

by

David Paul Courts
Lieutenant, United States Navy
B.S., University of North Carolina, 1966

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1973

Library
Naval Postgraduate School
Monterey, California 93940

ABSTRACT

A system for experimenting with semantic representations is developed. The existing REL System is discussed and comparisons are made. The proposed system, written in LISP, is described. A method for extending the user's language is developed. Recommendations for further study are presented. Program listings and sample results are included in the appendices.

TABLE OF CONTENTS

I. INTRODUCTION -- 5

II. THE REL SYSTEM ------------------------------------- 9

A. REL ENGLISH - A GENERAL LANGUAGE ----------- 10

1. REL English Vocabulary ----------------- 10

2. REL English Grammar -------------------- 10

3. Features ------------------------------- 11

B. ADVANTAGES OF THE REL ENGLISH SYNTAX ------- 13

III. THE PARSER --------------------------------------- 14

A. RULES OF GRAMMAR --------------------------- 14

1. Categories ----------------------------- 15

2. Use of "ANY" --------------------------- 16

B. COMPATIBILITY WITH REL ENGLISH SYNTAX ------ 17

IV. THE SSE SYSTEM ------------------------------------ 18

A. IMPLEMENTATION OF FEATURES ----------------- 18

B. THE "CHK" PROCEDURE ------------------------ 19

C. THE "CHG" FUNCTION ------------------------- 19

D. DEFINING WORDS ----------------------------- 19

E. "ADDRULE" ---------------------------------- 23

V. USERS MANUAL -- 27

VI. RECOMMENDATIONS ----------------------------------- 29

VII. CONCLUSIONS -------------------------------------- 30

APPENDIX 1: PART OF SPEECH FEATURES ----------------- 31

APPENDIX 2: SYSTEM LISTING -------------------------- 33

APPENDIX 3: GRAMMAR RULES --------------------------- 49

APPENDIX 4: SAMPLE RUN ------------------------------ 77
LIST OF REFERENCES ------------------------------------ 79
INITIAL DISTRIBUTION LIST ------------------------------ 80
FORM DD 1473 -- 81

I. INTRODUCTION

Question answering systems provide interaction between humans and computers in a natural language. A question answering system will accept queries to a data base and make replies in a language closely approximating natural English. The main components of such a system are a syntax for the query language, a data base, and some executive functions.

Current question answering systems are limited in their linguistic and semantic capabilities. Even the most successful systems will not allow the use of extremely elaborate sentence construction or obscure or poetic phraseology. The forms in which data is represented in the data base are also restricted. In successful systems data is represented by the equivalent of lists and tables, and the semantic functions generally are composed of various counting and sorting operations. Other types of semantic functions, such as cause and effect and inductive reasoning operations, are not available.

The present question answering systems appear to have progressed significantly toward solving purely syntactic problems. However, in the area of semantic problems such as reasoning and data representation, these same question answering systems appear to work at a very primitive level.

There exists a need for a system which would allow researchers to experiment with the semantic representation

of data. This report describes one such system, the Skeletal Semantic Experimentation System (SSE). The SSE system is so called because it provides a skeleton system (syntax, parser, and a partial executive system) to which the user must add desired semantic procedures and a data base.

Such a system as SSE would be required to have the following characteristics:

a) It must have an English syntax. The syntax must represent a major portion of natural English and be comprehensive enough to contend with problems which are purely syntactic in nature. For example the system must recognize that "A BOY" is an acceptable phrase while "A BOYS" is not.

The syntax must have a minimum of semantic preconceptions. It is not desired that the system have defined meanings for words. For example the system should accept the phrase

"DOG DAYS"

without making reference to a four legged animal.

b) The system should use a parser which is compatible with the syntax and is reasonably flexible and efficient.

c) The system must be transportable. It must be machine independent, relieving the experimenter of the task of modifing the system for his configuration.

In the past, for a researcher to experiment with data representations, he has had to expend considerable time and effort redoing work that had been previously accomplished.

Rather than concentrate his efforts purely on the data representation methods, a researcher has had to generate a syntax to represent natural English as well as a parser and control programs. The purpose of this project was to supply the researcher with a transportable system, written in LISP 1.5, [2], that would remove the requirement to generate a new syntax and parser. The system was developed by combining the well documented English syntax of the Rapidly Extensible Language System of B. H. Dostert and F. B. Thompson [3,4,5] with the General Purpose Parser of E. W. Merriam [7].

Dostert and Thompson's REL system, which has a parser built into it, is written in assembly language and is therefore totally dependent upon the type machine in which it resides. There are two choices open to an individual wishing to use a portion or all of the REL system. He may wish to use the entire system on a machine with which it is compatible or he may reprogram the desired portions of the REL system. The second alternative was chosen.

It is realized that programming in assembly language may be more efficient and when one is developing a production system this fact must be considered. However, the goal of SSE is not production but rather a transportable system that can be used by researchers at different computer facilities. For this purpose a higher level language is more appropriate.

SSE takes a sentence, parses it according to the REL English syntax, and returns the sentence in a diagrammed format. The returned sentence is then in a form that the

archer can use. He can apply his semantic procedures
e may wish to extract information from the sentence
use it for further experimentation.

A. REL ENGLISH - A GENERAL LANGUAGE

As a general language, REL English encompasses a considerable subset of natural English. In addition, REL English has the capability of performing elementary mathematical functions. However, REL English will not accept all of the grammatical constructions of ordinary English. Colloquial and extremely elaborate constructions are not part of REL English.

1. REL English Vocabulary

The vocabulary for REL English is limited to "little" words such as, "who", "which", "and", and "before". These "little" words are used in the definitions of the REL English syntax. The user then adds his own vocabulary to the system.

2. REL English Grammar

At present the grammar for REL English contains in excess of 350 rules [3] which are based on the centrality of the verb. The verb is considered to be the fulcrum of the sentence, and accumulates noun phrase modifiers, adverbial modifiers and auxiliaries about it. The parts of speech for REL English are as follows:

REL ENGLISH PARTS OF SPEECH:

NP	noun phrase	John, boy, locations of boys
VP	verb phrase	give, will give, give books
CV	copula verb	(i.e., be and have verbs)
NU	number	
AT	adverb of time	before June 1967, next year

CP	comparative	greater than, equal to
TD	time displacement	3 days, June
DI	digit	3
DS	digit string	345.
LO	locative	in Boston, where John lived [5].

Using the syntax for REL English, a considerable variety of natural English constructions can be interpreted. These constructions include complex verb structures, relative clauses, complex noun phrases, and conjunctions (both noun and sentence).

3. Features

The single concept of REL English which results in the successful evaluation of a syntactic structure, be it sentence or phrase, is the concept of "features." The idea of features is not a new one. It has been used before to subcategorize parts of speech, to identify, for example, a noun not just as a noun but as a plural noun or a possessive noun . The concept of features was implicit in a paper by G. N. Harman [6] and was discussed by Chomsky in *Aspects of the Theory of Syntax* [1].

The use of features in REL English results in a hierarchical structure of syntactic requirements. This ordering of syntactic requirements helps to control syntactic ambiguity and prevent ungrammatical constructions. For example, "some the boy" would not be accepted because quantifiers can not be combined with a noun which has had

the "definite" feature set due to the presence of the definite article.

Each part of speech is assigned a class of features which can be used with that part of speech. During the parsing of a sentence, the feature list for a given word or phrase will be compared with the features required by the grammar rule in question. If the features match then the rule can be applied. Appendix 1 lists the features for each part of speech with an example for each feature.

Features are also used to determine the best syntactic grouping when several groupings are acceptable. Dostert and Thompson find that grouping best which minimizes the number of accesses to the data base. For example, consider the phrase:

John's son who lives in Boston.

Of the two possible combinations:

a) John's (son who lives in Boston)
b) (John's son) who lives in Boston

(b) would be the best grouping since John is likely to have fewer sons than there are sons who live in Boston [4]. The number of accesses to the data base that the system makes is an important consideration for the REL system and other production question answering systems, since excessive accesses to a disk file can greatly reduce response time.

B. ADVANTAGES OF THE REL ENGLISH SYNTAX

The REL English syntax is well defined and documented. Sample sessions provided by Dostert and Thompson [5] show that the REL system works well, even with complicated sentence structures. REL English appears virtually free of semantic assumptions with the following exception: In defining a noun, REL English allows for three cases:

a) a name, such as John or book;
b) a class, such as firl;
c) a relation, such as sister or location.

In a natural language system certain purely syntactic problems develop regardless of the type of data representation being used. The example of the distinction between

A BOY and A BOYS

is a case in point. REL English with its concept of features does a most adequate job of handling these syntax problems.

But the most important advantage of the REL English syntax is that its documentation allows one to take advantage of the work that has been done and continue experimentation.

III. THE PARSER

To provide the required parsing operation, a slightly modified version of E. W. Merriam's General Purpose Parser was used [7]. The parser, written in LISP 1.5, takes a list of symbols, where each symbol stands for a word or phrase, and a list of grammar rules to be used in parsing the string. A symbol may be an atomic symbol, in which case it is a part of speech, or it may be a list structure. If it is a list, then the CAR (first element) of the list will be used as the part of speech by the parser.

In Merriam's parser a separate copy of the parsing rules is passed with each input string. In SSE the syntax rules are returned to "PARSE" as the result of a call to the function "RULES". In both versions the second argument for "PARSE" is a list of atomic symbols representing the sentence to be parsed.

A. RULES OF GRAMMAR

As in Merriam's version of the parser, the grammar rules consist of two parts. There is a left and a right side separated by a dot. Each of the grammar rules takes the form:

```
(( categories ) . f-name ).
```

III. THE PARSER

To provide the required parsing operation, a slightly modified version of E. W. Merriam's General Purpose Parser was used [7]. The parser, written in LISP 1.5, takes a list of symbols, where each symbol stands for a word or phrase, and a list of grammar rules to be used in parsing the string. A symbol may be an atomic symbol, in which case it is a part of speech, or it may be a list structure. If it is a list, then the CAR (first element) of the list will be used as the part of speech by the parser.

In Merriam's parser a separate copy of the parsing rules is passed with each input string. In SSE the syntax rules are returned to "PARSE" as the result of a call to the function "RULES". In both versions the second argument for "PARSE" is a list of atomic symbols representing the sentence to be parsed.

A. RULES OF GRAMMAR

As in Merriam's version of the parser, the grammar rules consist of two parts. There is a left and a right side separated by a dot. Each of the grammar rules takes the form:

((categories) . f-name).

1. Categories

The left side of a grammar rule is a list of categories. A category is an atomic symbol that will be compared with the original input string and future strings generated by the parsing procedure. If, during a comparison, a group of symbols in the input string match, by number and position, the symbols in the list of categories, then the function whose name is the right side of the rule is called. A list of the symbols that were matched in the input string is passed to the called function as an argument.

Consider the input string

```
(T H E  B O Y S).
```

Later in this report the method by which a word such as BOY can be defined to the system will be shown. Suffice it for now to say that BOY would be recognized as a noun phrase and the input string would look like

```
(T H E  (NP NAME BOY) S ).
```

Recall at this point that since the input string contains a list, the CAR of that list, "NP," will be used when compared with the grammar rule

```
(( NP S) . PLURAL).
```

In the grammar rule the list of categories is (NP S). If the input string contains the atom "NP" followed by the atom "S", then the function "PLURAL" will be called. The

list ((NP NAME BOY) S), from the input string, will be passed to "PLURAL" as an argument.

The value returned to the parser by the called function must be in form of a list. For example, the purpose of the function "PLURAL" is to build a new noun phrase which is the plural of the noun phrase in the argument. It must also turn on the plural feature and return the new noun phrase. Therefore, PLURAL would return

```
((NP NAME BOYS PLF +))
```

to the parser. This list will then be used for further parsing of the input string which now has the form

```
(T H E  (NP NAME BOYS PLF +))
```

2. Use of "ANY"

One of the significant features of Merriam's parser is the use of the word "ANY". If the word "ANY" appears in the list of categories, then any symbol occupying the corresponding position in the input string will be accepted. For example, both strings

```
1)  ( J O H N  :=NAME)
2)  (M A R Y  :=NAME)
```

would result in a call to the function "P1" when compared with the rule

```
(( ANY :=NAME) . P1).
```

As will be seen, this feature plays a major part in the users ability to define new words to the existing system.

B. COMPATIBILITY WITH REL ENGLISH SYNTAX

To make the Merriam parser and the REL English syntax compatible, little modification was required. It was necessary to make the REL English parts of speech atoms. A phrase is built of the CONS of the atomic part of speech and the description list for that phrase. For example, the word "DOG" would be represented as a list, the CAR of which would be the atom "NP" for noun phrase. The description list of the phrase would contain the attribute "NAME" and its value "DOG":

```
( NP NAME DOG )
```

IV. THE SSE SYSTEM

The Skeletal Semantic Experimentation System is a program which couples the modified version of the Merriam parser to the syntax of REL English. SSE was programmed in LISP for several reasons. First, LISP is a widely used programming language among researchers in artificial intelligence. Second, Merriam's parser is written in LISP and the REL English was in such a form that it could be implemented in LISP with relative ease. However, the primary consideration for using LISP was portability.

Because SSE uses no non-standard LISP functions, it should be compatible with any computer system having a LISP 1.5 compiler or interpreter.

A. IMPLEMENTATION OF FEATURES

In the Dostert and Thompson system the features of each part of speech are represented by a series of sixteen bits [5]. The Waterloo LISP system does not allow bit manipulation, so in SSE features and their values are placed on the description list of the phrase. The symbol for a feature is the three letter name of the feature listed in Appendix 1. Its value is either "+" if the feature is set or "-" if it is not. Since all parts of speech will not require the maximum of sixteen possible features in all cases, it was decided that initially a part of speech would contain no

features on its description list. In this manner it was implied that all possible features were set to "-" (off). This fact of implied feature values is taken into account in the "CHK" function.

B. THE "CHK" FUNCTION

The process of checking a given part of speech to insure that certain specified features have the desired values is performed by the function "CHK". The arguments for the function "CHK" are the phrase in question and a list of features, with their corresponding values, that are to be checked. The "CHK" function then looks on the description list of the phrase for each feature in the given list. If the result of a comparison is true then the next feature is checked. If the result if false, then the function will return "NIL", indicating that the features did not match.

C. THE "CHG" FUNCTION

Assuming that the result of the "CHK" function is true, the new part of speech to be returned to the parser may require that certain features be either added or modified. Modifing the description list is the role of "CHG". "CHG" accepts as arguments a phrase and a list of features which are to be added to its description list, or whose values are to be changed from their present value.

D. DEFINING WORDS

Words can be added to the user's lexicon as one of four types:

a)	NAME	Mary, dog, blonde
b)	CLASS	girl
c)	RELATION	location, mother
d)	VERB	enroll, arrive

The first three types result in a noun phrase being formed. The last type returns a verb phrase. A word is added by calling the parser using the format

```
( DEF : W O R D :=type )
```

where "type" is one of the four types listed above. To understand the procedure for defining words it is necessary to understand several of the parsing rules which use the word "ANY". The rules

```
((ANY :=NAME) . NME)
((ANY :=CLASS) .NME)
((ANY :=RELATION) . REL)
((ANY :=VERB) . VRB)
```

produce very similar results so that consideration of one of the rules will be sufficient to describe all four. Consider then the rule

```
((ANY :=NAME) . NME)
```

and an example definition

```
( DEF : J O H N :=NAME ).
```

When called, the function "NME" receives as arguments a list consisting of two atoms. The first call to "NME" would result in the list (N :=NAME) being passed as arguments.

In "NME" a check is made to insure that "N" is not a part of speech. If it were, then "NME" would return NIL and ignore the part of speech "N". However, since N is not a part of speech, the assumption is made that "N" is part of a word that is being defined. The list (N) is then appended to the value of the "NAME" on the description list of the atom ":=NAME" and the list ((:=NAME NAME (N))) is returned to the parser. At this point the example definition would be

(DEF : J O H (:=NAME NAME (N)))

and "NME" would be called again.

The arguments for this call would be (H (:=NAME NAME (N))). When the list (H) was appended to the value of "NAME" on the description list, the new value would be the list (H N). This process of collection letters would continue until the definition example had the form

(DEF : (:=NAME NAME (J O H N))).

With this call to "NME" the first atom in the list of arguments will be ":" and a different procedure will be followed. At this point it is assumed that all the letters of the new word have been collected. Therefore the atom "N1" is CONSed to the list (:=NAME NAME (J O H N)) forming the dotted pair

```
( N1 . ( :=NAME NAME ( J O H N ) ))
```

which is returned to the parser.

It is at this point that "NME", "REL", and "VRB" differ. "REL" and "VRB" would return "R1" and "V1" respectively in place of "N1".

The example definition now has the form

```
( DEF ( N1 . ( :=NAME NAME ( J O H N ) ))).
```

One must now consider the final parsing rule which uses the word "ANY":

```
(( DEF ANY ) . DEF).
```

When called, the function DEF is passed a list containing two elements. The first will always be "DEF". However, the second element could be either ":" or the returned value from "NME" "REL" or "VRB". If the second element is ":" then it is assumed that all the letters of the word being defined have not been collected. Therefore, the function "DEF" returns NIL to the parser. If the second element is not ":" then it must be either (N1 . (:=NAME NAME (J O H N))) or one of its variations as noted above.

The function "DEF" sets the variable FN to N1 (which is the name of a predefined function in the system). The value of "NAME" on the description list of ":=NAME" is extracted. The two items are CONSed to form a new parsing rule

```
(( J O H N ) . N1).
```

Using the NCONC function of LISP the new parsing rule is added to the list of parsing rules named by the function "RULES". At this point the word "J O H N" has been defined to the system and is available to all future calls to the parser.

E. ADDRULE

As a researcher experiments with various forms of data and various syntactic constructions, he may find it necessary to add his own syntactic rules to the system. The researcher may encounter a situation not previously covered, or he may wish to modify the order in which various syntactic groupings are being considered. For whatever reason the experimenter wishes to add a new syntactic rule, "ADDRULE" provides an effective method of so doing.

"ADDRULE" accepts as an argument a single list of the form

```
(( parser categories => part of speech to be returned)
   ( CHECK: features and values that must be true)
   ( SET: features and values that are to be set for
          the returned part of speech )
   ( name of semantic operator to be called )).
```

An example would be

```
ADDRULE ((
        ( CV NP => NP )
```

```
( CHECK: (FHV -) * (POF -)(RLF -) )
( SET: 1 (FOJ +)(FPH +) )
( SEMNOP ) )).
```
[1]

For this rule to apply, the part of speech, copula verb, must have the FHV feature turned off and the noun phrase must have the POF and RLF features off. The symbol "*" distinguishes which features are to apply to which part of speech. In the above example, had the "*" been missing, then "ADDRULE" would have assumed that all three features listed would apply to the first part of speech listed (i.e., CV).

The number following "SET:" indicates the "number" of the part of speech whose feature list is to be copied onto the description list of the returning value. In the above example the "1" indicates that a copy of the feature list of "CV" is to be placed on the description list of "VP". Had the number been "0", then the features of neither of the input parts of speech (i.e., CV or NP) would have been copied to "VP". Thus by implication, all the features on "VP" would have been off.

The list of features following the number are those features which are to be changed on the description list of the returned value. Thus from the above example it can be

---

[1] This format is a literal translation of the format used in Reference 5.

seen that "VP" will have at least the FOJ and FPH features turned on and the FHV feature turned off (since it was off on CV).

"SEMNOP" is the name of a function which performs no operations. This indicates that there are no semantic operations required for this particular grammar rule.

Appendix 2 lists the procedures for the system. With the procedures described above (i.e., the parser, the definition functions and "ADDRULE") one will also note representative procedures for the REL English Syntax, such as plural, possessive and part participle. These rules were not added to the system with the use of "ADDRULE" but were coded separately.

Appendix 3 lists the major portion of the REL English syntax. Each rule is in the format for entry into the system via the "ADDRULE" procedure.

One of the sentences used to check the system is listed in Appendix 4. The sentence was

(IS THERE A HARVARD BOY WHOSE SISTER ATTENDS YALE).

Having parsed the sentence, the system identified the sentence as a subjective verb phrase (FSJ) having a singular subject (FSI). It was also noted that the sentence is a question(FQT) containing an intransitive verb (FVI). After a successful parse of the sentence, the part of speech will contain the complete diagrammed sentence as the value of "NAME" on its description list. For the example listed in

Appendix 4, the value of "NAME" is

(IS THERE (A ((HARVARD BOY) WHOSE SISTER (ATTENDS
YALE)))).

This diagrammed sentence is in a form that the researcher can use to readily extract pertinent information for use in his experimentation with data representations.

V. USERS MANUAL

To facilitate use of the SSE system the user's requirements have been kept to a minimum. Appendices 2 and 3 contain the complete listing of the system. There are three possible inputs that the user can make to the system. He can define words, add syntax rules, or parse a sentence.

To define a word the user would use the following format

```
PARSE ((
      ( DEF : W O R D :=type ) )).
```

The word "type" would be replaced by

1) NAME
2) CLASS
3) RELATION
4) VERB

depending on the type of word to be defined. It should be noted that each symbol between "DEF" and " :=type" (i.e., :, W, O, R, and D) is an atomic symbol.

The format to be used to add a syntactic rule to the system was covered under "ADDRULE". Numerous examples of that format can be found in Appendix 3.

To parse a sentence the following format would be used

```
PARSE ((
      ( sentence ) ))
```

In this format each character in the sentence must be represented as an atomic symbol. For example

```
    PARSE ((
          ( W H E R E  I S  J O H N ) ))
```

VI. RECOMMENDATIONS

In developing the system to its present state, it was found that the full implications of several of the "machine semantic procedures" alluded to in the REL documentation were not clear. It is therefore recommended that further study be directed toward the development and implementation of these semantic procedures.

During the parsing of a sentence it would be reasonable to assume that the major portion of time spent determining the admissibility of a syntax rule would be spent checking features. For example there are fifteen syntax rules of the form

(NP VP = VP)

that are used to add subjects to a verb phrase. Therefore on the average there are seven rules that must be checked before finding the correct rule to apply. From the authors of the REL system it was learned that an average of six features must be checked for each rule. Thus to find the one syntax rule that allows the adding of a subject to a verb phrase one must check forty-two features. That is forty-two searches of a description list. By adding a small number of bit manipulation functions to LISP, more than one feature could be checked at a time. Features could be implemented using the LISP capability for octal representation, and the checking function could be performed by a masking operation.

VII. CONCLUSIONS

SSE can be a valuable tool to the researcher in semantic representations. The system is based on a comprehensive syntax and permits the parsing of complicated English sentences. The system returns these same sentences in a format which makes pertinent information readily accessible to the experimenter. Finally, because the system is written in LISP it is transportable to any computer system offering a LISP 1.5 complier or interpreter.

APPENDIX 1

NP FEATURES		EXAMPLE
PLF	plural	boys
POF	possessive	boy's
DTF	determiner	the boy
QNF	quantifier	all boys
APF	adjectural	male parent
PMF	"of" phrase	sister of John
PSF	possessive modifed	John's sister
RCF	relative clause modified	boy who lived in Boston
RLF	relation	sister, location
INF	individual	John, Boston
NAF	number argument	average
NVF	number valued	age, income
NAS	addition	income + capital loss
NMD	multiplication	income * tax rate
NXP	exponentiation	income **2
ANF	animate	John, girl

VP FEATURES		
FPH	phrasal, prevents further morphemic modification	
FSI	singular subject only	gives the book
FPP	past participle	arrived
FIF	infinitive	
FPA	passive	was given
FQT	question transformation	will John come

FIN	initial noun phrase	John gave books
FVI	intransitive verb	sleep
FSJ	subject	John gave books
FAG	agentive	John gave
FOJ	objective	books were given
FAT	time modified	arrived in June 1960
FDA	dative	gave to Mary
FAI	instrumental	opened with a key
FGE	genative	received from Mary
FAL	locative	arrived in Boston

CV FEATURES

Same as VP with the exception of FIN.

Replace FIN with

FHV "have" verb

NU FEATURES

NAS	addition or subtraction	$3+4$
NMD	multiplication or division	$3*4$
NXP	exponentiation	3^4
DTF	determiner	the age of John

APPENDIX 2

SYSTEM LISTING

```
*         GENERAL PURPOSE PARSER

DEFINE ((

 (PARSE (LAMBDA (ARGLIST) (PROGN
         (PAR2 (RULES NIL) (CAR ARGLIST)
              (SERCH (CDR ARGLIST) (QUOTE CAT) (QUOTE SYS1))
              (SERCH (CDR ARGLIST) (QUOTE OUTPUT) NIL)) )))

 (SERCH (LAMBDA (ARGLST ARG DEF) (COND
         ((NULL ARGLST) DEF)
         ((EQUAL (CAAR ARGLST) ARG) (CDAR ARGLST))
         (T (SERCH (CDR ARGLST) ARG DEF)) )))

 (PAR2 (LAMBDA (GRP STP2 CATPROG OUT) (PROGN
         (CSETQ TMP (PARS1 STP2 NIL))
         (COND
              ((NULL OUT) (ONLY TMP))
              (T TMP)) )))

 (ONLY (LAMBDA (RES) (COND
         ((NULL RES) NIL)
         ((NULL (CDAR RES)) (CONS (CAAR RES) (ONLY (CDR RES))))
         (T (ONLY (CDR RES))) )))

 (PARS1 (LAMBDA (STP END) (COND
         ((NULL (CDR STP)) (ADDLIST (LIST (CONS (CAR STP) END))))
         (T (ADDLIST (LIST (CONS (CAR STP) (PARS1 (CDR STP) END))))) )))

 (ADDLIST (LAMBDA (STA) (PROG (APG ATMP)
         (SETQ APG GRP)
         (SETQ ATMP (CATPROG (CAAR STA)))
      L1 (COND
              ((NULL APG) (RETURN STA))
              ((OR (EQUAL (CAAAR APG) (QUOTE ANY))
                   (EQUAL (CAAAR APG) ATMP) )
                   (MAKELIST (CDAAR APG) (CDAR APG) (CDAR STA) STA
                        (LIST (CAAR STA))) ) )
         (SETQ APG (CDR APG))
         (GO L1) )))

 (MAKELIST (LAMBDA (GR1M PGM STM LSTM ARGS) (COND
         ((NULL GR1M) (COND
```

```
                      ((NULL (CSETQ TEMP (PGM ARGS))) NIL)
                      (T (NCONC LSTM (PARS1 TEMP STM))) ) )
       ((NULL STM) NIL)
       (T (PROGN
           (COND
               ((EQUAL (CAR GR1M) (QUOTE ANY)) (MAKELIST (CDR GR1M)
               PGM (CDAR STM) LSTM (APPEND ARGS (LIST (CAAR STM)))))
               ((EQUAL (CAR GR1M) (CATPROG (CAAR STM))) (MAKELIST
                 (CDR GR1M) PGM (CDAR STM) LSTM (APPEND ARGS (LIST
                 (CAAR STM)))))
               (T NIL))
           (COND
               ((NULL (CDR STM)) NIL)
               (T (MAKELIST GR1M PGM (CDR STM) LSTM ARGS))) )) )))

 (CDAAR (LAMBDA (Q) (CDR (CAAR Q))))

 (SYS1 (LAMBDA (X) (COND ((ATOM X) X) (T (CAR X)) )))

*************       END  OF  PARSER                ************
```

```
*         SSE FUNCTIONS

 (SEMNOP (LAMBDA (X) NIL))

* MAKES AN ATOM FROM A GIVEN LIST

 (MAKEATOM (LAMBDA (X) (PROG (Y)
          (SETQ Y (COPY X))
          (CLEARBUFF)
   LOOP (COND ((NULL Y) (RETURN (MKATOM))))
          (PACK (CAR Y))
          (SETQ Y (CDR Y))
          (GO LOOP) )))

* CHECKS THE FEATURE LIST OF APART OF SPEECH

 (CHK (LAMBDA (X FEA)(PROG (TEST Y)
          (SETQ TEST FEA)
   LOOP   (COND
          ((NULL TEST) (RETURN T)))
          (SETQ Y (GET X (CAAR TEST)))
          (COND
          ((EQUAL Y NIL)(SETQ Y (QUOTE -))))
          (COND
          ((NULL (EQUAL (CADAR TEST) Y)) (RETURN NIL)))
          (SETQ TEST (CDR TEST))
          (GO LOOP) )))

* CHANGES THE VALUES OF FEATURES ON THE FEATURE LIST

 (CHG (LAMBDA (X FEAT)(PROG (TEST)
          (SETQ TEST FEAT)
   LOOP1 (COND ((NULL TEST) (RETURN (QUOTE T))))
          (PUT X (CAAR TEST)(CADAR TEST))
          (SETQ TEST (CDR TEST))
          (GO LOCP1) )))

* THE NEXT EIGHT FUNCTIONS ARE USED TO DEFINE A NAME, CLASS
* OR RELATICN NCUN; OR A VERB.

 (STUFF (LAMBDA (X Z)(PROG (FEA Y)
          (SETQ Y (COPY X))
          (SETQ FEA Z)
   LOOP2 (COND ((NULL FEA) (GO LOOP3        )))
          (PUT Y (CAAR FEA) (CADAR FEA))
          (SETQ FEA (CDR FEA))
          (GO LOOP2)
   LOOP3 (PUT Y (QUOTE NAME) X)
```

```
        (RETURN Y) )))

(NME (LAMBDA (X) (PROG (Y Z)
        (SETQ Y (CADR X))
        (COND
             ((EQUAL (CAR X) (QUOTE :)) (GO DONE))
             ((NULL (GET (CAR X) (QUOTE NAME))) (GO SKIP)))
        (RETURN NIL)
   SKIP (COND
             ((NULL (GET Y (QUOTE NAME)))
                  (PUT Y (QUOTE NAME)(LIST (CAR X))))
             (T (PUT Y (QUOTE NAME)(APPEND (LIST (CAR X))
                  (GET Y (QUOTE NAME))))))
        (RETURN (LIST Y))
   DONE (SETQ Z (CDR Y))
        (REMPROP Y (QUOTE NAME))
        (RETURN (LIST (CONS (QUOTE N1) Z))) )))

(REL (LAMBDA (X) (PROG (Y Z)
        (SETQ Y (CADR X))
        (COND
             ((EQUAL (CAR X) (QUOTE :)) (GO DONE))
             ((NULL (GET (CAR X) (QUOTE NAME))) (GO SKIP)))
        (RETURN NIL)
   SKIP (COND
             ((NULL (GET Y (QUOTE NAME)))
                  (PUT Y (QUOTE NAME)(LIST (CAR X))))
             (T (PUT Y (QUOTE NAME)(APPEND (LIST (CAR X))
                  (GET Y (QUOTE NAME))))))
        (RETURN (LIST Y))
   DCNE (SETQ Z (CDR Y))
        (REMPROP Y (QUOTE NAME))
        (RETURN (LIST (CONS (QUCTE R1) Z))) )))

(VRB (LAMBDA (X) (PRCG (Y Z)
        (SETQ Y (CADR X))
        (COND
             ((EQUAL (CAR X) (QUOTE :)) (GO DONE))
             ((NULL (GET (CAR X) (QUOTE NAME))) (GO SKIP)))
        (RETURN NIL)
   SKIP (CCND
             ((NULL (GET Y (QUOTE NAME)))
                  (PUT Y (QUOTE NAME)(LIST (CAR X))))
             (T (PUT Y (QUOTE NAME)(APPEND (LIST (CAR X))
                  (GET Y (QUOTE NAME))))))
        (RETURN (LIST Y))
   DONE (SETQ Z (CDR Y))
        (REMPROP Y (QUOTE NAME))
```

```
        (RETURN (LIST (CONS (QUOTE V1) Z))) )))

(DEF (LAMBDA (X) (PROG (ARGS FN)
        (COND ((EQUAL (CADR X) (QUOTE :)) (RETURN NIL)))
        (SETQ FN (CAADR X))
        (SETQ ARGS (GET (CADR X)(QUOTE NAME)))
        (NCONC (CADADDADDR (QUOTE RULES)) (LIST (CONS ARGS FN)))
        (RETURN NIL) )))

(N1 (LAMBDA (X)(PROG (Y Z)
        (SETQ Z NIL)
        (SETQ Y (CONS (QUOTE NP)(CDR (STUFF (MAKEATOM X) Z))))
        (RETURN (LIST Y)) )))

(R1 (LAMBDA (X) (PROG (Y Z)
        (SETQ Z (QUOTE ((RLF +))))
        (SETQ Y (CONS (QUOTE NP)(CDR (STUFF (MAKEATOM X) Z))))
        (RETURN (LIST Y)) )))

(V1 (LAMBDA (X)(PROG (Y Z)
        (SETQ Z NIL)
        (SETQ Y (CONS (QUOTE VP) (CDR (STUFF (MAKEATOM X) Z))))
        (RETURN (LIST Y)) )))

* ADDS NEW SYNTAX RULES TO EXISTING RULES

(ADDRULE (LAMBDA (X) (PROG (ARGS FN ARGS1 RTN FEA1 FEA2 FEA3 NUM FEAT
                SEM PROGRAM Y)

* GENERATES A UNIQUE FUNCTION NAME

        (SETQ FN (GENSYM))
        (SETQ ARGS (CAR X))
        (SETQ Y (COPY (CDR X)))
    LOOP (COND
            ((EQUAL (CAR ARGS) (QUOTE =>)) (GO AD1))
            ((ATOM ARGS1) (SETQ ARGS1 (LIST (CAR ARGS))))
            (T (SETQ ARGS1 (APPEND ARGS1 (LIST (CAR ARGS))))))
        (SETQ ARGS (CDR ARGS))
        (GO LOOP)

* ADDS THE NEW PARSING RULE TO "RULES"

    AD1 (NCONC (CADADDADDR (QUOTE RULES)) (LIST (CONS ARGS1 FN)))
        (SETQ RTN (CADR ARGS))
        (SETQ ARGS (CAR Y))
        (SETQ Y (CDR Y))
        (COND ((NULL (EQUAL (CAR ARGS) (QUOTE CHECK:))) (GO AD2)))
```

```
* COLLECTS THE FEATURES THAT ARE TO BE CHECKED

   LOOP1 (SETQ ARGS (CDR ARGS))
         (COND ((NULL ARGS) (GO CHK3)))
         (COND
              ((EQUAL (CAR ARGS) (QUOTE *)) (GO CHK1))
              ((ATOM FEA1) (SETQ FEA1 (LIST (CAR ARGS))))
              (T (SETQ FEA1 (APPEND FEA1 (LIST (CAR ARGS))))))
         (GO LOOP1)
    CHK1 (SETQ ARGS (CDR ARGS))
         (COND ((NULL ARGS) (GO CHK3)))
         (COND
              ((EQUAL (CAR ARGS) (QUOTE *)) (GO CH2))
              ((ATOM FEA2) (SETQ FEA2 (LIST (CAR ARGS))))
              (T (SETQ FEA2 (APPEND FEA2 (LIST (CAR ARGS))))))
         (GO CHK1)
     CH2 (SETQ ARGS (CDR ARGS))
         (COND ((NULL ARGS) (GO CH3)))
         (COND
              ((ATOM FEA3) (SETQ FEA3 (LIST (CAR ARGS))))
              (T (SETQ FEA3 (APPEND FEA3 (LIST (CAR ARGS))))))
         (GO CH2)
    CHK3 (SETQ ARGS (CAR Y))
         (SETQ Y (CDR Y))

* COLLECTS THE FEATURES THAT ARE TO BE CHANGED

     AD2 (COND ((NULL (EQUAL (CAR ARGS) (QUOTE SET:))) (GO AD3)))
         (SETQ NUM (CADR ARGS))
         (SETQ ARGS (CDDR ARGS))
   LOOP2 (COND ((NULL ARGS ) (GO SET1)))
         (COND
              ((ATOM FEAT) (SETQ FEAT (LIST (CAR ARGS))))
              (T (SETQ FEAT (APPEND FEAT (LIST (CAR ARGS))))))
         (SETQ ARGS (CDR ARGS))
         (GO LOOP2)
    SET1 (SETQ ARGS (CAR Y))

* SETS SEM TO THE SEMANTIC PROCEDURE THAT IS TO BE CALLED

     AD3 (SETQ SEM (CAR ARGS))
         (COND ((OR (EQUAL NUM 0) (NULL NUM)) (SETQ RTN (CONS RTN (CDR
              (STUFF (MAKEATOM ARGS1) NIL))))))
         (SETQ PROGRAM (QUOTE

* THE GENERAL FUNCTION
```

```
(FN (LAMBDA (X) (PROG (Y Z N M RTNV FEAC1 FEAC2 FEAC3)
        (SETQ M 1)
        (SETQ N NUM)
        (SETQ FEAC1 (QUOTE FEA1))
        (SETQ FEAC2 (QUOTE FEA2))
        (SETQ FEAC3 (QUCTE FEA3))
        (SETQ Y (COPY X))
    LUP (COND ((NULL Y) (GO OUT)))
        (SETQ Z (CAR Y))
        (SETQ Y (CDR Y))
        (COND ((ATOM Z) (GO LUP)))
        (COND ((NULL (EQUAL N M)) (GO STEP)))
        (SETQ RTNV (COPY Z))
        (RPLACA RTNV (QUOTE RTN))
   STEP (SETQ M (ADD1 M))
        (COND ((NULL (CHK Z FEAC1)) (RETURN NIL)))
        (SETQ FEAC1 FEAC2)
        (SETQ FEAC2 FEAC3)
        (COND ((NULL (EQUAL FEAC1 NIL)) (GO LUP)))
    OUT (COND ((EQUAL N 0) (SETQ RTNV (QUOTE RTN))))
        (COND ((NULL (NULL (QUOTE FEAT))) (CHG RTNV (QUOTE FEAT))))
        (REMPROP RTNV (QUOTE NAME))
        (SETQ M NIL)
        (SETQ Y (COPY X))
   LUP1 (COND ((NULL Y) (GO QUIT)))
        (SETQ Z (CAR Y))
        (SETQ Y (CDR Y))
        (COND ((ATOM Z) (GO ATM)))
        (COND ((NULL M) (GO JUMP)))
        (COND
             ((NULL (GET RTNV (QUOTE NAME))) (PUT RTNV (QUOTE NAME)
                  (APPEND (LIST (MAKEATOM M))
                       (LIST ( GET Z (QUOTE NAME))))))
             (T (PUT RTNV (QUOTE NAME) (APPEND (GET RTNV (QUOTE NAME))
                  (APPEND (LIST (MAKEATOM M))(LIST (GET Z (QUOTE NAME)
                       )))))))
        (SETQ M NIL)
        (GO LUP1)
   JUMP (COND
             ((NULL (GET RTNV (QUOTE NAME))) (PUT RTNV (QUOTE NAME)
                  (LIST (GET Z (QUOTE NAME)))))
             (T (PUT RTNV (QUOTE NAME) (APPEND (GET RTNV (QUOTE NAME))
             (LIST (GET Z (QUOTE NAME)))))))
        (GO LUP1)
    ATM (COND
             ((ATOM M) (SETQ M (LIST Z)))
             (T (SETQ M (APPEND M (LIST Z)))))
        (GO LUP1)
```

```
    QUIT (COND ((NULL (NULL M)) (PUT RTNV (QUOTE NAME) (MAKEATOM M))))
         (SEM NIL)
         (RETURN (LIST RTNV)) )))

         ) )

* SUBSTITUTE THE TRUE VALUES FOR THE DUMMY VARIABLES

         (SETQ PROGRAM (SUBST FN (QUOTE FN) PROGRAM))
         (SETQ PROGRAM (SUBST RTN (QUOTE RTN) PROGRAM))
         (SETQ PROGRAM (SUBST FEA1 (QUOTE FEA1) PROGRAM))
         (SETQ PROGRAM (SUBST FEA2 (QUOTE FEA2) PROGRAM))
         (SETQ PROGRAM (SUBST FEA3 (QUOTE FEA3) PROGRAM))
         (SETQ PROGRAM (SUBST NUM (QUOTE NUM) PROGRAM))
         (SETQ·PROGRAM (SUBST FEAT (QUOTE FEAT) PROGRAM))
         (SETQ PROGRAM (SUBST SEM (QUOTE SEM) PRCGRAM))

* MAKE THE NEW FUNCTICN A PART OF THE EXISTING FUNCTIONS

         (DEFINE (LIST PROGRAM))
         (RETURN (QUCTE ENTERED)) )))

* LIST OF RULES FOR THE PARSER

 (RULES (LAMBDA (X)(QUOTE (
         ((DEF ANY). DEF)
         ((ANY :=NAME). NME)
         ((ANY :=CLASS). NME)
         ((ANY :=RELATION). REL)
         ((ANY :=VERB). VRB)
         ((NP S). PLURAL)
         ((NP ' S). POSSESSIVE)
         ((NP '). POSSESSIVE)
         ((VP S). VERBSING)
         ((VP D). PASTPART)
         ((VP E D). PASTPART)
         ((NP NP). ADJMOD)
         ((NP NP). POSMOD)
         ((NP OF NP). PREP)
         ((NP WHO VP). WHOCLS)
         ((NP COMMA WHO VP). WHOCOM)
         ((NP THAT VP). THATCLS)
         ((NP COMMA THAT VP). THATCOM)
         ((NP WHICH VP). WHICHCLS)
         ((NP COMMA WHICH VP). WHICHCOM)
         ((NP WHOM VP). WHOMCLS)
         ((NP CCMMA WHCM VP). WHOMCOM)
         ((NP BY WHOM VP). BYWHMCLS)
```

```
       ((NP CCMMA BY WHCM VP). BYWHMCOM)
       ((NP WHOSE NP VP). WHOSECLS)
       ((NP COMMA WHOSE NP VP). WHOSECOM)

* NEW PARSING RULES WILL BE ENTERED HERE

                   ) )))

* THE FOLLOWING SYNTAX RULES DO NOT USE "ADDRULE".
* THEY WERE CODED SEPERATLY

 (PLURAL (LAMBDA (X) (PROG (FEA FEAT Y)

* THESE ARE THE FEATURES WHICH MUST BE SET FOR THIS RULE TO APPLY

       (SETQ FEA (QUOTE ((APF -)(DTF -)(PLF -)(PMF -)(POF -)(PSF -)
           (RCF -)(QNF -)(INF -)))))

* THIS IS THE FEATURE WHICH WILL BE SET IF THE RULE APPLIES

       (SETQ FEAT (QUOTE ((PLF +))))
       (COND ((NULL (CHK (CAR X) FEA)) (RETURN NIL)))
       (SETQ Y (COPY (CAR X)))
       (CHG Y FEAT)

* USING THE NAME OF THE NP, MAKE ITS PLURAL AND PLACE IT ON THE
* PROPERTY-LIST OF THE NEW NP

       (PUT Y (QUOTE NAME)(MAKEATOM (APPEND (LIST
           (GET Y (QUCTE NAME))) (CDR X)))))

* RETURN THE NEW NP

       (RETURN (LIST Y)) )))

 (POSSESSIVE (LAMBDA (X) (PROG (FEAT Y)
       (SETQ FEAT (QUOTE ((POF +))))
       (COND ((NULL (CHK (CAR X) (QUOTE ((POF -))))) (RETURN NIL)))
       (SETQ Y (COPY (CAR X)))
       (CHG Y FEAT)
       (PUT Y (QUOTE NAME)(MAKEATOM (APPEND (LIST
           (GET Y (QUOTE NAME))) (CDR X)))))
       (RETURN (LIST Y)) )))

 (VERBSING (LAMBDA (X) (PROG (FEAT Y)
       (SETQ FEAT (QUOTE ((FPH +)(FSI +))))
       (COND ((NULL (CHK (CAR X) (QUOTE ((FPH -))))) (RETURN NIL)))
       (SETQ Y (COPY (CAR X)))
```

```
          (CHG Y FEAT)
          (PUT Y (QUOTE NAME)(MAKEATOM (APPEND (LIST
               (GET Y (QUOTE NAME)) (CDR X))))
          (RETURN (LIST Y)) )))

(PASTPART (LAMBDA (X) (PROG (FEAT Y)
          (SETQ FEAT (QUOTE ((FPH +)(FPP +))))
          (COND ((NULL (CHK (CAR X) (QUOTE ((FPH -))))) (RETURN NIL)))
          (SETQ Y (COPY (CAR X)))
          (CHG Y FEAT)
          (PUT Y (QUOTE NAME)(MAKEATOM (APPEND (LIST
               (GET Y (QUOTE NAME)) (CDR X))))
          (RETURN (LIST Y)) )))

(ADJMOD   (LAMBDA (X) (PROG (FEAT Y)
          (SETQ FEAT (QUOTE ((APF +))))
          (SETQ Y (COPY (CADR X)))
          (COND
               ((CHK (CAR X) (QUOTE ((RLF -)(POF -)(DTF -)(QNF -)(APF -)
                                     (PMF -)(PSF -)(RCF -))))
                    (COND
                         ((CHK (CADR X) (QUOTE ((RLF -)(POF -)(DTF -)
                          (QNF -)(PMF -)(PSF -)(RCF -)))) (GO ADJ1))))
               ((CHK (CAR X) (QUOTE ((RLF -)(POF -)(DTF -)(QNF -)(APF -)
                                     (PMF -)(PSF -)(RCF -))))
                    (COND
                         ((CHK (CADR X) (QUOTE ((RLF +)(POF -)(DTF -)
                          (PMF -)(PSF -)(RCF -)))) (GO ADJ1))))
               ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -)(APF -)(PMF -)
                                     (PSF -)(RCF -)(NAF -)(NVF -))))
                    (COND
                         ((CHK (CADR X) (QUOTE ((RLF -)(POF -)(DTF -)
                          (QNF -)(PSF -)(RCF -)))) (GO ADJ1))))
               ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -)(PMF -)(RCF -)
                                     (NAF +))))
                    (COND
                         ((CHK (CADR X) (QUOTE ((RLF +)(POF -)(DTF -)
                          (PSF -)(RCF -)(PMF -)(NVF -)))) (GO ADJ1))))
               ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -)(PMF -)(RCF -)
                                     (NVF -))))
                    (COND
                         ((CHK (CADR X) (QUOTE ((RLF +)(POF -)(DTF -)
                          (PMF -)(PSF -)(RCF -)(NAF -)))) (GO ADJ2)))))
          (RETURN NIL)
     ADJ1 (CHG Y FEAT)
          (GO ADJ3)
     ADJ2 (CHG Y (QUOTE ((PSF +))))
     ADJ3 (PUT Y (QUOTE NAME) (LIST (GET (CAR X)(QUOTE NAME))
```

```
                                  (GET (CADR X)(QUOTE NAME))))
        (RETURN (LIST Y)) )))

(POSMOD (LAMBDA (X) (PROG (Y)
        (SETQ Y (COPY (CADR X)))
        (COND
             ((CHK (CAR X) (QUOTE ((RLF -)(POF +)(DTF -)(PMF -)(RCF -)
                                   )))
                  (CCND
                       ((CHK (CADR X) (QUOTE ((RLF -)(POF -)(DTF -)
                        (QNF -)(PMF -)(PSF -)(RCF -)))) (GO POS1))))
             ((CHK (CAR X) (QUOTE ((RLF -)(POF +)(DTF -)(PMF -)(RCF -)
                                   )))
                  (COND
                       ((CHK (CADR X) (QUOTE ((RLF +)(POF -)(DTF -)
                        (PMF -)(PSF -)(RCF -)(NAF -)(NVF -))))
                                   (GO POS2))))
             ((CHK (CAR X) (QUOTE ((RLF +)(POF +)(DTF -)(PMF -)(RCF -)
                                   )))
                  (COND
                       ((CHK (CADR X) (QUOTE ((RLF -)(POF -)(DTF -)
                        (QNF -)(PMF -)(PSF -)(RCF -)))) (GO POS3)))))
        (RETURN NIL)
   POS1 (CHG Y (QUOTE ((PSF +))))
        (GO POS4)
   POS2 (CHG Y (QUOTE ((PSF +)(RLF -))))
        (GO POS4)
   POS3 (CHG Y (QUOTE ((PSF +)(RLF +))))
   POS4 (PUT Y (QUOTE NAME) (LIST (GET (CAR X)(QUOTE NAME))
                                  (GET (CADR X)(QUOTE NAME))))
        (RETURN (LIST Y)) )))

(PREP (LAMBDA (X) (PROG (Y)
        (SETQ Y (COPY (CAR X)))
        (COND
             ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -)(PSF -)(RCF -)
                                   (PMF -)(NAF -)(NVF -))))
                  (COND
                       ((CHK (CADDR X) (QUOTE ((RLF -)(POF -))))
                                   (GO PREP1))))
             ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -)(PSF -)(RCF -)
                                   (PMF -))))
                  (COND
                       ((CHK (CADDR X) (QUOTE ((RLF +)(DTF -)(POF -))))
                                   (GO PREP2))))
             ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -)(PSF -)(PMF -)
                                   (RCF +)(NAF -)(NVF -))))
                  (COND
```

```
                       ((CHK (CADDR X) (QUOTE ((RLF -)(POF -)(RCF +))))
                                   (GO PREP1))))
             ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -)(PSF -)(RCF +)
                                   (PMF -))))
                  (CCND
                       ((CHK (CADDR X) (QUOTE ((RLF +)(POF -)(DTF -)
                                   (RCF +)))) (GO PREP2))))
             ((CHK (CAR X) (QUOTE ((RLF +)(PCF -)(DTF -)(PSF -)(RCF -)
                                   (PMF -)(NAF +))))
                  (COND
                       ((CHK (CADDR X) (QUOTE ((RLF +)(POF -)(DTF -)
                                   (NAF -)(NVF +)))) (GO PREP3))))
             ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -)(PSF -)(RCF +)
                                   (PMF -)(NAF +))))
                  (COND
                       ((CHK (CADDR X) (QUOTE ((RLF +)(POF -)(DTF -)
                                (RCF +)(NAF -)(NVF +)))) (GO PREP3))))
             ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -)(PSF -)(RCF -)
                                   (PMF -)(NAF +))))
                  (COND
                       ((CHK (CADDR X) (QUOTE ((RLF +)(POF -)(DTF +)
                                   (NAF +)(NVF +)))) (GO PREP4))))
             ((CHK (CAR X) (QUOTE ((RLF +)(PCF -)(DTF -)(PSF -)(RCF +)
                                   (PMF -)(NAF +))))
                  (CCND
                       ((CHK (CADDR X) (QUOTE ((RLF +)(POF -)(DTF -)
                               (RCF +)(NAF +)(NVF +)))) (GO PREP4)))))
        (RETURN NIL)
  PREP1 (CHG Y (QUOTE ((PMF +)(RLF -))))
        (GO PREP4)
  PREP2 (CHG Y (QUOTE ((PMF +))))
        (GO PREP4)
  PREP3 (CHG Y (QUOTE ((NAF -))))
  PREP4 (PUT Y (QUOTE NAME) (LIST (GET (CAR X)(QUOTE NAME)) (QUOTE OF)
                                  (GET (CADDR X)(QUOTE NAME))))
        (RETURN (LIST Y)) )))

(WHOCLS (LAMBDA (X) (PROG (Y)
        (SETC Y (COPY (CAR X)))
        (COND
             ((CHK (CAR X) (QUOTE ((RLF -)(POF -)(DTF -)(QNF -)(PMF -)
                                   (RCF -)))) (GO WHO1))
             ((CHK (CAR X) (QUOTE ((RLF +)(PCF -)(DTF -)(PMF -)(RCF -)
                                   ))) (GO WHO1)))
        (RETURN NIL)
   WHO1 (CHG Y (QUOTE ((RCF +))))
        (PUT Y (QUOTE NAME) (LIST (GET (CAR X)(QUOTE NAME)) (QUOTE WHO)
                                  (GET (CADDR X)(QUOTE NAME))))
```

```
        (RETURN (LIST Y)) )))

(WHOCOM (LAMBDA (X) (PROG (Y)
        (SETQ Y (COPY (CAR X)))
        (COND
             ((CHK (CAR X) (QUOTE ((RLF -)(POF -)(DTF -)(QNF -))))
                                   (GO WH1))
             ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -)(PMF -)(RCF -)
                                   ))) (GO WH1)))
        (RETURN NIL)
    WH1 (CHG Y (QUOTE ((RCF +))))
        (PUT Y (QUOTE NAME) (LIST (GET (CAR X)(QUOTE NAME))
             (QUOTE $$', WHO') (GET (CADDDR X)(QUOTE NAME))))
        (RETURN (LIST Y)) )))

(THATCLS (LAMBDA (X) (PROG (Y)
        (COND
             ((CHK (CAR X) (QUOTE ((RLF -)(POF -)(DTF -)(QNF -)(PMF -)
                                   (RCF -)))) (GO THAT1))
        (SETQ Y (COPY (CAR X)))
             ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -)(PMF -)(RCF -)
                                   ))) (GO THAT1)))
        (RETURN NIL)
  THAT1 (CHG Y (QUOTE ((RCF +))))
        (PUT Y (QUOTE NAME) (LIST (GET (CAR X)(QUOTE NAME))(QUOTE THAT)
                                  (GET (CADDR X)(QUOTE NAME))))
        (RETURN (LIST Y)) )))

(THATCOM (LAMBDA (X) (PROG (Y)
        (SETQ Y (COPY (CAR X)))
        (COND
             ((CHK (CAR X) (QUOTE ((RLF -)(POF -)(DTF -)(QNF -))))
                                   (GO TH1))
             ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -)))) (GO TH1)))
        (RETURN NIL)
    TH1 (CHG Y (QUOTE ((RCF +))))
        (PUT Y (QUOTE NAME) (LIST (GET (CAR X)(QUOTE NAME))
             (QUOTE $$', THAT') (GET (CADDDR X)(QUOTE NAME))))
        (RETURN (LIST Y)) )))

(WHICHCLS (LAMBDA (X) (PROG (Y)
        (SETQ Y (COPY (CAR X)))
        (COND
             ((CHK (CAR X) (QUOTE ((RLF -)(POF -)(DTF -)(QNF -)(PMF -)
                                   (RCF -)))) (GO WHICH))
             ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -)(PMF -)(RCF -)
                                   ))) (GO WHICH)))
        (RETURN NIL)
```

```
  WHICH (CHG Y (QUOTE ((RCF +))))
        (PUT Y (QUOTE NAME) (LIST (GET (CAR X)(QUOTE NAME))
                    (QUOTE WHICH)   (GET (CADDR X)(QUOTE NAME))))
        (RETURN (LIST Y)) )))

(WHICHCOM (LAMBDA (X) (PROG (Y)
        (SETQ Y (COPY (CAR X)))
        (COND
             ((CHK (CAR X) (QUOTE ((RLF -)(POF -)(DTF -)(QNF -))))
                                    (GO WCH))
             ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -)))) (GO WCH)))
        (RETURN NIL)
    WCH (CHG Y (QUOTE ((RCF +))))
        (PUT Y (QUOTE NAME) (LIST (GET (CAR X)(QUOTE NAME))
             (QUOTE $$', WHICH') (GET (CADDDR X)(QUOTE NAME))))
        (RETURN (LIST Y)) )))

(WHOMCLS (LAMBDA (X) (PRCG (FEAT Y)
        (SETQ FEAT (QUOTE ((FPA -))))
        (SETQ Y (COPY (CAR X)))
        (COND
             ((CHK (CAR X) (QUOTE ((RLF -)(POF -)(DTF -)(QNF -)(PMF -)
                                    (RCF -))))
                  (CCND
                       ((CHK (CADDR X) FEAT) (GO WHOM))))
             ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -)(PMF -)(RCF -)
                                    )))
                  (COND
                       ((CHK (CADDR X) FEAT) (GC WHOM)))))
        (RETURN NIL)
   WHOM (CHG Y (QUOTE ((RCF +))))
        (PUT Y (QUOTE NAME) (LIST (GET (CAR X)(QUOTE NAME))(QUOTE WHOM)
                                    (GET (CADDR X)(QUOTE NAME))))
        (RETURN (LIST Y)) )))

(WHOMCOM (LAMBDA (X) (PRCG (FEAT Y)
        (SETQ FEAT (QUOTE ((FPA -))))
        (SETC Y (COPY (CAR X)))
        (COND
             ((CHK (CAR X) (QUOTE ((RLF -)(POF -)(DTF -)(QNF -))))
                  (CCND
                       ((CHK (CADDDR X) FEAT) (GO WHM))))
             ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -))))
                  (CCND
                       ((CHK (CADDCR X) FEAT) (GO WHM)))))
        (RETURN NIL)
    WHM (CHG Y (QUOTE ((RCF +))))
        (PUT Y (QUOTE NAME) (LIST (GET (CAR X)(QUCTE NAME))
```

```
             (QUOTE $$', WHOM') (GET (CADDDR X)(QUOTE NAME))))
        (RETURN (LIST Y)) )))

(BYWHMCLS (LAMBDA (X) (PROG (FEAT Y)
        (SETQ FEAT (QUOTE ((FPA +))))
        (SETQ Y (COPY (CAR X)))
        (COND
             ((CHK (CAR X) (QUOTE ((RLF -)(POF -)(DTF -)(QNF -)(PMF -)
                                   (RCF -))))
                  (COND
                       ((CHK (CADDDR X) FEAT) (GO BY))))
             ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -)(PMF -)(RCF -)
                                   )))
                  (COND
                       ((CHK (CADDDR X) FEAT) (GO BY)))))
        (RETURN NIL)
     BY (CHG Y (QUOTE ((RCF +))))
        (PUT Y (QUOTE NAME) (LIST (GET (CAR X)(QUOTE NAME))
             (QUOTE $$'BY WHOM') (GET (CADDDR X)(QUOTE NAME))))
        (RETURN (LIST Y)) )))

(BYWHMCOM (LAMBDA (X) (PROG (FEAT Y)
        (SETQ FEAT (QUOTE ((FPA +))))
        (SETQ Y (COPY (CAR X)))
        (COND
             ((CHK (CAR X) (QUOTE ((RLF -)(POF -)(DTF -)(QNF -))))
                  (COND
                       ((CHK (CADDDDR X) FEAT) (GO BY1))))
             ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -))))
                  (COND
                       ((CHK (CADDDDR X) FEAT) (GO BY1)))))
        (RETURN NIL)
    BY1 (CHG Y (QUOTE ((RCF +))))
        (PUT Y (QUOTE NAME) (LIST (GET (CAR X)(QUOTE NAME))
             (QUOTE $$', BY WHOM') (GET (CADDDDR X)(QUOTE NAME))))
        (RETURN (LIST Y)) )))

(WHOSECLS (LAMBDA (X) (PROG (FEAT Y)
        (SETQ FEAT (QUOTE ((RLF +)(POF -)(DTF -))))
        (SETQ Y (COPY (CAR X)))
        (COND
             ((CHK (CAR X) (QUOTE ((RLF -)(POF -)(DTF -)(QNF -)(PMF -)
                                   (RCF -))))
                  (COND
                       ((CHK (CADDR X) FEAT) (GO WHOSE))))
             ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -)(PMF -)(RCF -)
                                   )))
                  (COND
```

```
                        ((CHK (CADDR X) FEAT) (GO WHOSE)))))
            (RETURN NIL)
      WHOSE (CHG Y (QUOTE ((RCF +))))
            (PUT Y (QUOTE NAME) (LIST (GET (CAR X)(QUOTE NAME))
                 (QUOTE WHOSE)(GET (CADDR X)(QUOTE NAME))(GET (CADDDR X)
                 (QUOTE NAME))))
            (RETURN (LIST Y)) )))

   (WHOSECOM (LAMBDA (X) (PROG (FEAT Y)
            (SETQ FEAT (QUOTE ((RLF +)(POF -)(DTF -))))
            (SETQ Y (COPY (CAR X)))
            (COND
                 ((CHK (CAR X) (QUOTE ((RLF -)(POF -)(DTF -)(QNF -))))
                      (COND
                           ((CHK (CADDDR X) FEAT) (GO WHS1))))
                 ((CHK (CAR X) (QUOTE ((RLF +)(POF -)(DTF -))))
                      (COND
                           ((CHK (CADDDR X) FEAT) (GO WHS1)))))
            (RETURN NIL)
       WHS1 (CHG Y (QUOTE ((RCF +))))
            (PUT Y (QUOTE NAME) (LIST (GET (CAR X)(QUOTE NAME))
                 (QUOTE $$', WHOSE')(GET (CADDDR X)(QUOTE NAME))
                 (GET (CADDCDR X)(QUOTE NAME))))
            (RETURN (LIST Y)) )))

   ))
```

APPENDIX 3

GRAMMAR RULES

```
ADDRULE ( (
          (VP NP => NP)
          (CHECK: (FPP +)(FSJ -)(FAG -)(FOJ -)(FDA -)(FAI -)(FAL -)
                 (FGE -)(FAT -) * (RLF -)(POF -)(DTF -)(QNF -)(PMF -)
                 (PSF -)(RCF -) )
          (SET: 2 (APF +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (W H A T VP => NP)
          (CHECK: (FSJ -)(FQT -) )
          (SET: 0 (DTF +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (W H A T VP => NP)
          (CHECK: (FSJ +)(FOJ -)(FQT -) )
          (SET: 0 (DTF +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (T H E NP => NP)
          (CHECK: (RLF -)(POF -)(DTF -)(QNF -)(INF -) )
          (SET: 1 (DTF +)(INF +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (T H E NP => NP)
          (CHECK: (RLF -)(POF -)(DTF -)(QNF -)(INF +) )
          (SET: 1 (DTF +)(RLF -) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (T H E NP => NP)
          (CHECK: (RLF +)(POF -)(DTF -) )
          (SET: 1 (DTF +)(RLF +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (A NP => NP)
          (CHECK: (RLF -)(POF -)(DTF -)(QNF -) )
          (SET: 1 (DTF +)(RLF -) )
          (SEMNOP)
```

```
        ) )
ADCRULE ( (
        (A NP => NP)
        (CHECK: (RLF +)(POF -)(DTF -) )
        (SET: 1 (DTF +)(RLF +) )
        (SEMNOP)
        ) )
ADCRULE ( (
        (E V E R Y NP => NP)
        (CHECK: (RLF-)(PCF -)(DTF -)(QNF -)(PSF -)(PLF -) )
        (SET: 1 (QNF +)(RLF -) )
        (SEMNOP)
        ) )
ADCRULE ( (
        (A L L NP => NP)
        (CHECK: (RLF -)(PCF -)(QNF -)(PSF -)(PLF +) )
        (SET: 1 (QNF +)(PLF -)(RLF -) )
        (SEMNOP)
        ) )
ADDRULE ( (
        (N O NP => NP)
        (CHECK: (RLF -)(POF -)(DTF -)(QNF -) )
        (SET: 1 (QNF +)(FLF -)(RLF -) )
        (SEMNOP)
        ) )
ADDRULE ( (
        (S O M E NP => NP)
        (CHECK: (RLF -)(POF -)(DTF -)(QNF -)(PSF -) )
        (SET: 1 (QNF +)(PLF -)(RLF -) )
        (SEMNOP)
        ) )
ADDRULE ( (
        (N O T   A L L NP => NP)
        (CHECK: (RLF -)(POF -)(QNF -)(PLF +) )
        (SET: 1 (QNF +)(PLF -)(RLF -) )
        (SEMNOP)
        ) )
ADDRULE ( (
        (E A C H NP => NP)
        (CHECK: (RLF -)(POF -)(DTF -)(QNF -)(PSF -)(PLF -) )
        (SET: 1 (QNF +)(RLF -) )
        (SEMNOP)
        ) )
ADDRULE ( (
        (W H I C H NP => NP)
        (CHECK: (RLF -)(POF -)(DTF -)(QNF -)(PSF -) )
        (SET: 1 (QNF +)(PLF -)(RLF -) )
        (SEMNOP)
```

```
                ) )
ADDRULE ( (
                (W H A T NP => NP)
                (CHECK: (RLF -)(POF -)(DTF -)(QNF -)(PSF -) )
                (SET: 1 (QNF +)(POF -)(RLF -) )
                (SEMNOP)
                ) )
ADDRULE ( (
                (H O W   M A N Y NP => NP)
                (CHECK: (RLF -)(POF -)(DTF -)(QNF -)(PSF -)(PLF +) )
                (SET: 1 (QNF +)(PLF -)(RLF -) )
                (SEMNOP)
                ) )
ADDRULE ( (
                (N U M B E R => NP)
                (SET 0 (RLF +)(NVF +)(NAF-) )
                (SEMNOP)
                ) )
ADDRULE ( (
                (S U M => NP)
                (SET: 0 (RLF+)(NVF +)(NAF +) )
                (SEMNOP)
                ) )
ADDRULE ( (
                (M A X I M U M => NP)
                (SET: 0 (RLF +)(NVF +)(NAF +) )
                (SEMNOP)
                ) )
ADDRULE ( (
                (M I N I M U M => NP)
                (SET: 0 (RLF +)(NVF +)(NAF +) )
                (SEMNOP)
                ) )
ADDRULE ( (
                (A V E R A G E => NP)
                (SET: 0 (RLF +)(NVF +)(NAF +) )
                (SEMNOP)
                ) )
ADDRULE ( (
                (NP + NP => NP)
                (CHECK: (RLF +)(POF -)(DTF -)(NAF -)(NVF +) * (RLF +)(POF -)
                    (NAF -)(NVF +)(NAS -) )
                (SET: 1 (NAS +) )
                (SEMNOP)
                ) )
ADDRULE ( (
                (NP + NP => NP)
                (CHECK: (RLF +)(POF -)(DTF -)(NAF +)(NVF +) * (RLF +)(POF -)
```

```
              (NAF +)(NVF +)(NAS -) )
         (SET: 1 (NAS +) )
         (SEMNOP)
         ) )
ADDRULE ( (
         (NP - NP => NP)
         (CHECK: (RLF +)(POF -)(DTF -)(NAF -)(NVF +) * (RLF +)(POF -)
              (NAF -)(NVF +)(NAS -) )
         (SET: 1 (NAS +) )
         (SEMNOP)
         ) )
ADDRULE ( (
         (NP - NP => NP)
         (CHECK: (RLF +)(POF -)(DTF -)(NAF +)(NVF +) * (RLF +)(POF -)
              (NAF +)(NVF +)(NAS -) )
         (SET: 1 (NAS +) )
         (SEMNOP)
         ) )
ADDRULE ( (
         (NP + NU => NP)
         (CHECK: (RLF +)(POF -)(DTF -)(NVF +) * (NAS -) )
         (SET: 1 (NAS +) )
         (SEMNOP)
         ) )
ADDRULE ( (
         (NP - NU => NP)
         (CHECK: (RLF +)(POF -)(DTF -)(NVF +) * (NAS -) )
         (SET: 1 (NAS +) )
         (SEMNOP)
         ) )
ADDRULE ( (
         (NU + NP => NP)
         (CHECK: * (RLF +)(POF -)( DTF -)(NVF +)(NAS _) )
         (SET: 2 (NAS +) )
         (SEMNOP)
         ) )
ADDRULE ( (
         (NU - NP => NP)
         (CHECK: * (RLF +)(POF -)( DTF -)(NVF +)(NAS _) )
         (SET: 2 (NAS +) )
         (SEMNOP)
         ) )
ADDRULE ( (
         (+ NP => NP)
         ((CHECK: (RLF +)(POF -)(NVF +)(NAS -) )
         (SET: 1 (NAS +) )
         (SEMNOP)
         ) )
```

```
ADDRULE ( (
          (- NP => NP)
          ((CHECK: (RLF +)(POF -)(NVF +)(NAS -) )
          (SET: 1 (NAS +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (NP * NP => NP)
          (CHECK: (RLF +)(POF -)(DTF -)(NAF -)(NVF +)(NAS -) * (RLF +)
                (POF -)(NAF -)(NVF +)(NAS -)(NMD -) )
          (SET: 1 (NMD +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (NP * NP => NP)
          (CHECK: (RLF +)(POF -)(DTF -)(NAF +)(NVF +)(NAS -) * (RLF +)
                (POF -)(NAF +)(NVF +)(NAS -)(NMD -) )
          (SET: 1 (NMD +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (NP / NP => NP)
          (CHECK: (RLF +)(POF -)(DTF -)(NAF -)(NVF +)(NAS -) * (RLF +)
                (POF -)(NAF -)(NVF +)(NAS -)(NMD -) )
          (SET: 1 (NMD +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (NP / NP => NP)
          (CHECK: (RLF +)(POF -)(DTF -)(NAF +)(NVF +)(NAS -) * (RLF +)
                (POF -)(NAF +)(NVF +)(NAS -)(NMD -) )
          (SET: 1 (NMD +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (NP * NU => NP)
          (CHECK: (RLF +)(DTF -)(POF -)(NVF +)(NAS -) * (NAS -)(NMD -) )
          (SET: 1 (NMD +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (NP / NU => NP)
          (CHECK: (RLF +)(DTF -)(POF -)(NVF +)(NAS -) * (NAS -)(NMD -) )
          (SET: 1 (NMD +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (NU * NP => NP)
```

```
          (CHECK: (NAS -) * (RLF +)(POF -)(NVF +)(NAS -)(NMD -) )
          (SET: 2 (NMD +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (NU / NP => NP)
          (CHECK: (NAS -) * (RLF +)(POF -)(NVF +)(NAS -)(NMD -) )
          (SET: 2 (NMD +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (NU * * NP => NP)
          (CHECK: (RLF +)(POF -)(DTF -)(NVF +)(NAS -)(NMD -) * (RLF +)
                (POF -)(NVF +)(NAS -)(NMD -)(NXP -) )
          (SET: 1 (NXP +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (NP * * NU => NP)
          (CHECK: (RLF +)(POF -)(DTF -)( NVF +)(NAS -)(NMD -) * (NAS -)
                (NMD -)(NXP -) )
          (SET: 1 (NXP +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (NU * * NP => NP)
          (CHECK: (NAS -)(NMD -) * (RLF +)(POF -)(NVF +)(NAS -)(NMD -)
                (NXP -) )
          (SET: 2 (NXP +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          ( (NP) => NP)
          (CHECK: (RLF +)(POF -)(NVF +) )
          (SET: 1 (NAS -)(NMD -)(NXP -)(PMF -)(POF -)(PSF -)(RCF -)
                (APF -) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (T H E NU => NU)
          (CHECK: (DTF -) )
          (SET: 1 (DTF +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (NP NP => NU)
          (CHECK: (RLF -)(POF -)(DTF -)(QNF -)(PMF -)(RCF -) * (RLF +)
                (PCF -)(DTF -)(PLF -)(PMF -)(PSF -)(RCF -)(NAF -)(NVF +) )
```

```
            (SET: 0 (PLF -) )
            (SEMNOP)
            ) )
ADDRULE ( (
            (NP NP => NU)
            (CHECK: (RLF -)(POF -)(DTF -)(QNF -)(PMF -)(RCF -) * (RLF +)
                   (POF -)(DTF -)(PLF +)(PMF -)(PSF -)(RCF -)(NAF -)(NVF +) )
            (SET: 0 (PLF +) )
            (SEMNOP)
            ) )
ADDRULE ( (
            (NP NP => NU)
            (CHECK: (RLF -)(POF +)(DTF -)(PMF -)(RCF -) * (RLF +)(POF -)
                   (DTF -)(PMF -)(PSF -)(RCF -)(NAF -)(NVF +)(PLF -) )
            (SET: 0 (PLF -) )
            (SEMNOP)
            ) )
ADDRULE ( (
            (NP NP => NU)
            (CHECK: (RLF -)(POF +)(DTF -)(PMF -)(RCF -) * (RLF +)(POF -)
                   (DTF -)(PMF -)(PSF -)(RCF -)(NAF -)(NVF +)(PLF +) )
            (SET: 0 (PLF +) )
            (SEMNOP)
            ) )
ADDRULE ( (
            (NP O F NP => NU)
            (CHECK: (RLF +)(POF -)(DTF -)(PSF -)(PMF -)(RCF -)(PLF -)
                   (NAF -)(NVF +) * (POF -) )
            (SET: 0 (PLF -) )
            (SEMNOP)
            ) )
ADDRULE ( (
            (NP O F NP => NU)
            (CHECK: (RLF +)(POF -)(DTF -)(PSF -)(PMF -)(RCF +)(PLF -)
                   (NAF -)(NVF +) * (POF -)(RCF +) )
            (SET: 0 (PLF -) )
            (SEMNOP)
            ) )
ADDRULE ( (
            (NP O F NP => NU)
            (CHECK: (RLF +)(POF -)(DTF -)(PSF -)(PMF -)(RCF -)(PLF +)
                   (NAF -)(NVF +) * (POF -) )
            (SET: 0 (PLF +) )
            (SEMNOP)
            ) )
ADDRULE ( (
            (NP O F NP => NU)
            (CHECK: (RLF +)(POF -)(DTF -)(PSF -)(PMF -)(RCF +)(PLF +)
```

```
               (NAF -)(NVF +) * (POF -)(RCF +) )
          (SET: 0 (PLF +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (NP O F NU => NU)
          (CHECK: (RLF +)(POF -)(DTF -)(PSF -)(PMF -)(RCF -)(PLF -)
               (NAF +)(NVF +) * (POF -) )
          (SET: 0 (PLF -) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (NP O F NU => NU)
          (CHECK: (RLF +)(POF -)(DTF -)(PSF -)(PMF -)(RCF -)(PLF +)
               (NAF +)(NVF +) * (POF -) )
          (SET: 0 (POF +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (I T => NP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (H E => NP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (S H E => NP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (T H E Y => NP)
          (SET: 0 (PLF +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (I T S => NP)
          (SET 0 (POF +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (H I S => NP)
          (SET: 0 (POF +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (H E R => NP)
          (SET: 0 (POF +) )
```

```
           (SEMNOP)
           ) )
ADDRULE ( (
           (T H E I R => NP)
           (SET: 0 (POF +)(PLF +) )
           (SEMNOP)
           ) )
ADDRULE ( (
           (H I M => NP)
           (SEMNOP)
           ) )
ADDRULE ( (
           (T H E M => NP)
           (SET: 0 (PLF +) )
           (SEMNOP)
           ) )
ADDRULE ( (
           (VP NP => VP)
           (CHECK: (FVI -)(FSJ -)(FOJ -)(FAG -)(FDA -)(FGE -)(FAI -)
                 (FAL -)(FAT -) * (RLF -)(POF -) )
           (SET: 1 (FOJ +)(FPH +) )
           (SEMNOP)
           ) )
ADDRULE ( (
           (CV NP => VP)
           (CHECK: (FHV -) * (POF -)(RLF -) )
           (SET: 1 (FOJ +)(FPH +) )
           (SEMNOP)
           ) )
ADDRULE ( (
           (VP NP NP => VP)
           (CHECK: (FIF -)(FVI -)(FSJ -)(FAG -)(FOJ -)(FDA -)(FGE -)
                 (FAI -)(FAL -)(FAT -)(FPA -) * (RLF -)(POF -)(ANF +)
                 * (RLF -)(POF -) )
           (SET: 1 (FOJ +)(FDA +)(FPH +) )
           (SEMNOP)
           ) )
ADDRULE ( (
           (VP B Y NP => VP)
           (CHECK: (FVI -)(FSJ -)(FAG -)(FPA +) * (ANF +)(POF -)(RLF -) )
           (SET: 1 (FAG +)(FPA +)(FPH +) )
           (SEMNOP)
           ) )
ADDRULE ( (
           (VP B Y NP => VP)
          (CHECK: (FVI -)(FSJ -)(FAG -)(FPA -)(FPP +) * (ANF +)(POF -)
                 (RLF -) )
           (SET: 1 (FAG +)(FPA +)(FPH +) )
```

```
         (SEMNOP)
         ) )
ADDRULE ( (
         (VP B Y NP => VP)
         (CHECK: (FSJ -)(FAG -)(FAI -)(FPA +) * (ANF -)(POF -)(RLF -) )
         (SET: 1 (FAI +)(FPA +)(FPH +)(FAG +) )
         (SEMNOP)
         ) )
ADDRULE ( (
         (VP B Y NP => VP)
         (CHECK: (FSJ -)(FAG -)(FAI -)(FPP +) * (ANF -)(POF -)(RLF -) )
         (SET: 1 (FAI +)(FPA +)(FPH +) FAG +) )
         (SEMNOP)
         ) )
ADDRULE ( (
         (VP W I T H NP => VP)
         (CHECK: (FSJ -)(FAI -) * (ANF -)(POF -)(RLF -) )
         (SET: 1 (FAI +)(FPH +) )
         (SEMNOP)
         ) )
ADDRULE ( (
         (VP T C NP => VP)
         (CHECK: (FSJ -)(FDA -) * (POF -)(RLF -) )
         (SET: 1 (FDA +)(FPH +) )
         (SEMNOP)
         ) )
ADDRULE ( (
         (VP NU => VP)
         (CHECK: (FOJ -)(FSJ -) * )
         (SET: 1 (FOJ +)(FPH +) )
         (SEMNOP)
         ) )
ADDRULE ( (
         (CV NU => VP)
         (CHECK: (FHV -) * )
         (SET: 1 (FOJ +)(FPH +)(FVI +) )
         (SEMNOP)
         ) )
ADDRULE ( (
         (NP VP => VP)
         CHECK: (RLF -)(POF -)(ANF +) * (FSJ -)(FIN -)(FQT -)(FPA -)
                (FSI -)(FAT -)(FAG -) )
         (SET: 2 (FPH +)(FSJ +)(FAG +)(FIN +) )
         (SEMNOP)
         ) )
ADDRULE ( (
         (NP VP => VP)
         (CHECK: (RLF -)(POF -)(ANF +)(PLF -) * (FSJ -)(FIN -)(FQT -)
```

```
                 (FPA -)(FSI +)(FAT -)(FAG -) )
            (SET: 2 (FPH +)(FSJ +)(FAG +)(FIN +) )
            (SEMNOP)
            ) )
ADCRULE ( (
            (NP VP => VP)
            CHECK: (RLF -)(POF -)(ANF -) * (FSJ -)(FIN -)(FQT -)(FPA -)
                 (FSI -)(FAT -)(FAG -)(FAI -)(FOJ +) )
            (SET: 2 (FPH +)(FSJ +)(FAI +)(FIN +) )
            (SEMNOP)
            ) )
ADDRULE ( (
            (NP VP => VP)
            (CHECK: (RLF -)(POF -)(ANF -)(PLF -) * (FSJ -)(FIN -)(FQT -)
                 (FPA -)(FSI +)(FAT -)(FAG -)(FAI -)(FOJ +) )
            (SET: 2 (FPH +)(FSJ +)(FAI +)(FIN +) )
            (SEMNOP)
            ) )
ADCRULE ( (
            (NP VP => VP)
            (CHECK: (POF -)(RLF -)(ANF -) * (FSJ -)(FIN -)(FQT -)(FPA -)
                 (FSI -)(FAT -)(FAG -)(FAI -)(FOJ -) )
            (SET: 2 (FPH +)(FSJ +)(FOJ +)(FIN +) )
            (SEMNOP)
            ) )
ADCRULE ( (
            (NP VP => VP)
            (CHECK: (POF -)(RLF -)(ANF -)(PLF -) * (FSJ -)(FIN -)(FQT -)
                 (FPA -)(FSI +)(FAT -)(FAG -)(FAI -)(FOJ -) )
            (SET: 2 (FPH +)(FSJ +)(FCJ +)(FIN +) )
            (SEMNOP)
            ) )
ADCRULE ( (
            (NP VP => VP)
           (CHECK: (RLF -)(POF -)(ANF +) * (FSJ -)(FIN -)(FQT -)(FPA -)
                 (FSI -)(FAT -)(FAG -)(FPP +) )
            (SET: 2 (FPH +)(FSJ +)(FAG +)(FIN +)(FPP -) )
            (SEMNOP)
            ) )
ADDRULE ( (
            (NP VP => VP)
            (CHECK: (RLF -)(POF -)(ANF +)(PLF -) * (FSJ -)(FIN -)(FQT -)
                 (FPA -)(FSI +)(FAT -)(FAG -)(FPP +) )
            (SET: 2 (FPH +)(FSJ +)(FAG +)(FIN +)(FPP -) )
            (SEMNOP)
            ) )
ADCRULE ( (
            (NP VP => VP)
```

```
        CHECK: (RLF -)(POF -)(ANF -) * (FSJ -)(FIN -)(FQT -)(FPA -)
             (FSI -)(FAT -)(FAG -)(FAI -)(FOJ +)(FPP +) )
        (SET: 2 (FPH +)(FSJ +)(FAI +)(FIN +)(FPP -) )
        (SEMNOP)
        ) )
ADDRULE ( (
        (NP VP => VP)
        (CHECK: (RLF -)(POF -)(ANF -)(PLF -) * (FSJ -)(FIN -)(FQT -)
             (FPA -)(FSI +)(FAT -)(FAG -)(FAI -)(FOJ +) FPP +) )
        (SET: 2 (FPH +)(FSJ +)(FAI +)(FIN +)(FPP -) )
        (SEMNOP)
        ) )
ADDRULE ( (
        (NP VP => VP)
        (CHECK: (POF -)(RLF -)(ANF -) * (FSJ -)(FIN -)(FQT -)(FPA -)
             (FSI -)(FAT -)(FAG -)(FAI -)(FOJ -)(FPP +) )
        (SET: 2 (FPH +)(FSJ +)(FOJ +)(FIN +)(FPP -) )
        (SEMNOP)
        ) )
ADDRULE ( (
        (NP VP => VP)
        (CHECK: (POF -)(RLF -)(ANF -)(PLF -) * (FSJ -)(FIN -)(FQT -)
             (FPA -)(FSI +)(FAT -)(FAG -)(FAI -)(FOJ -)(FPP +) )
        (SET: 2 (FPH +)(FSJ +)(FOJ +)(FIN +)(FPP -) )
        (SEMNOP)
        ) )
ADDRULE ( (
        (NP VP => VP)
        (CHECK: (RLF -)(POF -) * (FSJ -)(FIN -)(FQT -)(FPA +)(FSI -)
             (FAT -)(FOJ -) )
        (SET: 2 (FPH +)(FSJ +)(FOJ +)(FIN +) )
        (SEMNOP)
        ) )
ADDRULE ( (
        (NP VP => VP)
        (CHECK: (RLF -)(POF -)(PLF -) * (FSJ -)(FIN -)(FQT -)(FPA +)
             (FSI +)(FAT -)(FOJ -) )
        (SET: 2 (FPH +)(FSJ +)(FOJ +)(FIN +) )
        (SEMNOP)
        ) )
ADDRULE ( (
        (NP VP => VP)
        (CHECK: (RLF -)(POF -) * (FSJ -)(FIN -)(FQT -)(FPA +)(FSI -)
             (FAT -)(FOJ +)(FDA -) )
        (SET: 2 (FPH +)(FSJ +)(FDA +)(FIN +) )
        (SEMNOP)
        ) )
ADDRULE ( (
```

```
          (NP VP => VP)
          (CHECK: (RLF -)(POF -)(PLF -) * (FSJ -)(FIN -)(FQT -)(FPA +)
               (FSI +)(FAT -)(FOJ +)(FDA -) )
          (SET: 2 (FPH +)(FSJ +)(FCA +)(FIN +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (NU VP => VP)
          (CHECK: * (FSJ -)(FIN -) )
          (SET: 2 (FPH +)(FSJ +)(FIN +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (CV VP => VP)
          (CHECK: (FHV -)(FPH -) * (FPP +)(FSJ -)(FAG -)(FOJ -)(FDA -)
               (FGE -)(FAI -)(FAL -)(FAT -)(FPA -)(FVI -) )
          (SET: 1 (FPH +)(FPA +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (CV VP => VP)
          (CHECK: (FHV -)(FPH -) * (FPP +)(FSJ +)(FVI -) )
          (SET: 1 (FPH +)(FPA +)(FQT +)(FSJ +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (CV VP => VP)
          (CHECK: (FHV +)(FPH -) * (FPP +)(FSJ -)(FAG -)(FOJ -)(FDA -)
               (FGE -)(FAI -)(FAL -)(FAT -)(FPA -)(FVI -) )
          (SET: 1 (FIN -)(FPH +) )
          (SEMNOP)
          ) )
ADCRULE ( (
          (CV VP => VP)
          (CHECK: (FHV +)(FPH -) * (FPP +)(FSJ +)(FPA -) )
          (SET: 1 (FPH +)(FSJ +)(FQT +)(FIN -) )
          (SEMNOP)
          ) )
ADCRULE ( (
          (CV VP => VP)
          (CHECK: (FHV +)(FPH -) * (FPP +)(FSJ -)(FAG -)(FOJ -)(FDA -)
               (FGE -)(FAI -)(FAL -)(FAT -)(FPA +)(FVI -) )
          (SET: 1 (FIN -)(FPH +)(FPA +) )
          (SEMNOP)
          ) )
ADCRULE ( (
          (CV VP => VP)
          (CHECK: (FHV +)(FPH -) * (FPP +)(FSJ +)(FPA +) )
```

```
                (SET: 1 (FPH +)(FSJ +)(FQT +)(FPA +)(FIN -) )
                (SEMNOP)
                ) )
ADDRULE ( (
                (CV VP => VP)
                (CHECK: (FHV -)(FPH -) * (FPP +)(FSJ -)(FAG -)(FOJ -)(FDA -)
                        (FGE -)(FAI -)(FAL -)(FAT -)(FVI +) )
                (SET: 1 (FPH +)(FPA -) )
                (SEMNOP)
                ) )
ADDRULE ( (
                (CV VP => VP)
                (CHECK: (FPH -) * (FPP +)(FSJ +)(FVI +) )
                (SET: 1 (FPH +)(FPA -)(FQT +)(FSJ +) )
                (SEMNOP)
                ) )
ADDRULE ( (
                (D O VP => VP)
                (CHECK: (FQT -)(FSJ +) )
                (SET: 1 (FPH +)(FIN -)(FQT +) )
                (SEMNOP)
                ) )
ADDRULE ( (
                (D O VP => VP)
                (CHECK: (FQT -)(FSJ -)(FAG -)(FOJ -)(FDA -)(FAI -)(FAL -)
                        (FGE -)(FAT -) )
                (SET: 1 (FPH +)(FIN -) )
                (SEMNOP)
                ) )
ADDRULE ( (
                (D O E S VP => VP)
                (CHECK: (FQT -)(FSJ +) )
                (SET: 1 (FPH +)(FIN -)(FQT +)(FSI +) )
                (SEMNOP)
                ) )
ADDRULE ( (
                (D O E S VP => VP)
                (CHECK: (FQT -)(FSJ -)(FAG -)(FOJ -)(FDA -)(FAI -)(FAL -)
                        (FGE -)(FAT -) )
                (SET: 1 (FPH +)(FIN -)(FSI +) )
                (SEMNOP)
                ) )
ADDRULE ( (
                (D I D VP => VP)
                (CHECK: (FQT -)(FSJ +) )
                (SET: 1 (FPH +)(FIN -)(FQT +) )
                (SEMNOP)
                ) )
```

```
ADDRULE ( (
          (D I D VP => VP)
          (CHECK: (FQT -)(FSJ -)(FAG -)(FOJ -)(FDA -)(FAI -)(FAL -)
                (FGE -)(FAT -) )
          (SET: 1 (FPH +)(FIN -) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (W I L L VP => VP)
          (CHECK: (FQT -)(FSJ +) )
          (SET: 1 (FPH +)(FIN -)(FQT +)(FIF -) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (W I L L VP => VP)
          (CHECK: (FQT -)(FSJ -)(FAG -)(FOJ -)(FDA -)(FAI -)(FAL -)
                (FGE -)(FAT -) )
          (SET: 1 (FPH +)(FIN -)(FIF -) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (D O N ' T VP => VP)
          (CHECK: (FQT -)(FSJ +) )
          (SET: 1 (FPH +)(FIN -)(FQT +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (D O N ' T VP => VP)
          (CHECK: (FQT -)(FSJ -)(FAG -)(FOJ -)(FDA -)(FAI -)(FAL -)
                (FGE -)(FAT -) )
          (SET: 1 (FPH +)(FIN -) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (D O E S N ' T VP => VP)
          (CHECK: (FQT -)(FSJ +) )
          (SET: 1 (FPH +)(FIN -)(FQT +)(FSI +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (D O E S N ' T VP => VP)
          (CHECK: (FQT -)(FSJ -)(FAG -)(FOJ -)(FDA -)(FAI -)(FAL -)
                (FGE -)(FAT -) )
          (SET: 1 (FPH +)(FIN -)(FSI +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (D I D N ' T VP => VP)
```

```
               (CHECK: (FQT -)(FSJ +) )
               (SET: 1 (FPH +)(FIN -)(FQT +) )
               (SEMNOP)
               ) )
ADDRULE ( (
               (D I D N ' T VP => VP)
               (CHECK: (FQT -)(FSJ -)(FAG -)(FOJ -)(FDA -)(FAI -)(FAL -)
                    (FGE -)(FAT -) )
               (SET: 1 (FPH +)(FIN -) )
               (SEMNOP)
               ) )
ADDRULE ( (
               (W O N ' T VP => VP)
               (CHECK: (FQT -)(FSJ +) )
               (SET: 1 (FPH +)(FIN -)(FQT +)(FIF -) )
               (SEMNOP)
               ) )
ADDRULE ( (
               (W O N ' T VP => VP)
               (CHECK: (FQT -)(FSJ -)(FAG -)(FOJ -)(FDA -)(FAI -)(FAL -)
                    (FGE -)(FAT -) )
               (SET: 1 (FPH +)(FIN -)(FIF -) )
               (SEMNOP)
               ) )
ADDRULE ( (
               (CV NP => CV)
               (CHECK: (FOJ -)(FSJ -)(FSI -) * (RLF -)(POF -) )
               (SET: 1 (FSJ +)(FQT +)(FPH +) )
               (SEMNOP)
               ) )
ADDRULE ( (
               (CV NP => CV)
               (CHECK: (FOJ -)(FSJ -)(FSI +) * (RLF -)(POF -)(PLF -) )
               (SET: 1 (FSJ +)(FQT +)(FPH +) )
               (SEMNOP)
               ) )
ADDRULE ( (
               (CV NP => CV)
               (CHECK: (FOJ -)(FSJ -)(FHV -) * )
               (SET: 1 (FSJ +)(FQT +)(FPH +) )
               (SEMNOP)
               ) )
ADDRULE ( (
               (NP VP => VP)
               (CHECK: (RLF -)(POF -)(QNF +) * (FQT +)(FSJ +)(FIN -)(FOJ -)
                    (FPA -)(FAT -) )
               (SET: 2 (FOJ +)(FIN +) )
               (SEMNOP)
```

```
               ) )
ADDRULE (  (
               (B Y NP VP  => VP)
               (CHECK: (RLF -)(POF -)(QNF +)(ANF +) * (FQT +)(FSJ +)(FIN -)
                    (FAG -)(FPA +)(FAT -) )
               (SET: 2 (FAG +) )
               (SEMNOP)
               ) )
ADDRULE (  (
               (B Y NP VP  => VP)
               (CHECK: (RLF -)(POF -)(QNF +)(ANF -) * (FQT +)(FSJ +)(FIN -)
                    (FAG -)(FAI -)(FPA -)(FAT -) )
               (SET: 2 (FAI +) )
               (SEMNOP)
               ) )
ADDRULE (  (
               (CV NP  => VP)
               (CHECK: (FHV +)(FPH -) * (RLF +)(POF -) )
               (SET: 1 (FPH +)(FIN -) )
               (SEMNOP)
               ) )
ADDRULE (  (
               (CV NP  => VP)
               (CHECK: (FHV +)(FQT +)(FSJ +) * (RLF +)(POF -) )
               (SET: 1 (FPH +)(FIN -) )
               (SEMNOP)
               ) )
ADDRULE (  (
               (N O T VP  => VP)
               (CHECK: (FSJ -)(FAG -)(FOJ -)(FDA -)(FGE -)(FAI -)(FAL -)
                    (FAT -)(FVI -) )
               (SET: 1 (FPH +) )
               (SEMNOP)
               ) )
ADDRULE (  (
               (CV N O T  => CV)
               (CHECK: (FPH -) )
               (SET: 1 (FPH +) )
               (SEMNOP)
               ) )
ADDRULE (  (
               (CV N ' T  => CV)
               (CHECK: (FPH -) )
               (SEMNOP)
               ) )
ADDRULE (  (
               (CV N O T  => CV)
               (CHECK: (FQT +)(FSJ +) )
```

```
                  (SET: 1 (FPH +) )
                  (SEMNOP)
                  ) )
ADDRULE ( (
                  (VP AT => VP)
                  (CHECK: (FAT -) * )
                  (SET: 1 (FAT +)(FPH +) )
                  (SEMNOP)
                  ) )
ADDRULE ( (
                  (W H E N VP => AT)
                  (CHECK: (FSJ +)(FPP -) )
                  (SEMNOP)
                  ) )
ADDRULE ( (
                  (W H E R E VP => LO)
                  (CHECK: (FSJ +)(FPP -) )
                  (SEMNOP)
                  ) )
ADDRULE ( (
                  (CV T H E R E NP => VP)
                  (CHECK: (FHV -)(FSJ -)(FPP -) * (POF -)(RLF -) )
                  (SET: 1 (FSJ +)(FPH +)(FQT +)(FVI +) )
                  (SEMNOP)
                  ) )
ADDRULE ( (
                  (A R E => CV)
                  (SEMNOP)
                  ) )
ADDRULE ( (
                  (B E => CV)
                  (SET: 0 (FIF +) )
                  (SEMNOP)
                  ) )
ADDRULE ( (
                  (W A S => CV)
                  (SET: 0 (FSI +) )
                  (SEMNOP)
                  ) )
ADDRULE ( (
                  (W E R E => CV)
                  (SEMNOP)
                  ) )
ADDRULE ( (
                  (B E E N => CV)
                  (SET: 0 (FPP +) )
                  (SEMNOP)
                  ) )
```

```
ADDRULE ( (
          (H A S => CV)
          (SET: 0 (FHV +)(FSI +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (H A V E => CV)
          (SET: 0 (FHV +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (H A D => CV)
          (SET: 0 (FHV +)(FPP +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (T H E R E CV NP => VP)
          (CHECK: (FHV -)(FSJ -) * (POF -)(RLF -) )
          (SET: 1 (FSJ +)(FPH +)(FVI +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (I S => CV)
          (SET: 0 (FSI +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (VP CP => VP)
          (SET: 1 (FPH +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (CV CP => VP)
          (SET: 1 (FPH +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (= => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (E Q U A L => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (E Q U A L   T C => CP)
          (SEMNOP)
          ) )
```

```
ADDRULE ( (
          (T H E   S A M E   A S => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (< => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (L E S S   T H A N => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (L O W E R   T H A N => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          ( S M A L L E R   T H A N => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (> => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (G R E A T E R   T H A N => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (H I G H E R   T H A N => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (L A R G E R   T H A N => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (M O R E   T H A N => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (O R   E Q U A L => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (O R   E Q U A L   T O => CP)
          (SEMNOP)
          ) )
```

```
ADDRULE ( (
          (< = => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (A S   H I G H   A S => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (A S   L A R G E   A S => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (A S   G R E A T   A S => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          ((> = => C P)
          (SEMNOP)
          ) )
ADDRULE ( (
          (A S   L O W   A S => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (A S   S M A L L   A S => CP)
          (SEMNOP)
          ) )
ADDRULE ( (
          (J A N U A R Y => TD)
          (SET: C (DAT +)(SNG +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (F E B R U A R Y => TD)
          (SET: O (DAT +)(SNG +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (M A R C H => TD)
          (SET: O (DAT +)(SNG +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (A P R I L => TD)
          (SET: O (DAT +)(SNG +) )
          (SEMNOP)
          ) )
```

```
ADDRULE ( (
          (M A Y => TD)
          (SET: 0 (DAT +)(SNG +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (J U N E => TD)
          (SET: 0 (DAT +)(SNG +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (J U L Y => TD)
          (SET: 0 (DAT +)(SNG +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (A U G U S T => TD)
          (SET: 0 (DAT +)(SNG +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (S E P T E M B E R => TD)
          (SET: 0 (DAT +)(SNG +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (O C T O B E R => TD)
          (SET: 0 (DAT +)(SNG +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (N O V E M B E R => TD)
          (SET: 0 (DAT +)(SNG +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (D E C E M B E R => TD)
          (SET: 0 (DAT +)(SNG +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (D A Y => TD)
          (SET: 0 (MAT +)(SNG +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (M O N T H => TD)
          (SET: 0 (MAT +)(SNG +) )
```

```
          (SEMNOP)
          ) )
ADDRULE ( (
          (Y E A R => TD)
          (SET: 0 (MAT +)(SNG +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (TD AT => AT)
          (CHECK: (MAT -)(DAT +)(PAT -)(SNG +) * (MAT +)(PAT -) )
          (SET: 2 (MAT -) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (TD , AT => AT)
          (CHECK: (DAT -)(MAT -)(PAT -) * (DAT +)(MAT +)(PAT -) )
          (SET: 2 (DAT -)(MAT -) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (T H I S TD => AT)
          (CHECK: (SNG +)(PAT -) )
          (SET: 0 )
          (SEMNOP)
          ) )
ADDRULE ( (
          (L A S T TD => AT)
          (CHECK: (SNG +)(PAT -) )
          (SET: 0 )
          (SEMNOP)
          ) )
ADDRULE ( (
          (N E X T TD => AT)
          (CHECK: (SNG +)(PAT -) )
          (SET: 0 )
          (SEMNOP)
          ) )
ADDRULE ( (
          (B E F O R E AT => AT)
          (CHECK: (PAT -) )
          (SET: 1 (PAT +) )
          (SEMNOP)
          ) )
ADDRULE ( (
          (A F T E R AT => AT)
          (CHECK: (PAT -) )
          (SET: 1 (PAT +) )
          (SEMNOP)
```

```
               ) )
ADDRULE     ( (
               (S I N C E AT => AT)
               (CHECK: (PAT -) )
               (SET: 1 (PAT +) )
               (SEMNOP)
               ) )
ADDRULE     ( (
               (I N AT => AT)
               (CHECK: (PAT -) )
               (SET: 1 )
               (SEMNOP)
               ) )
ADDRULE     ( (
               (I N TD => AT)
               (CHECK: (PAT -)(SNG -)(MAT +) )
               (SET: 0 (MAT -)(DAT -)(PAT +) )
               (SEMNOP)
               ) )
ADDRULE     ( (
               (I N TD => AT)
               (CHECK: (PAT -)(SNG +)(MAT +) )
               (SET: 0 (MAT -)(DAT -)(PAT +) )
               (SEMNOP)
               ) )
ADDRULE     ( (
               (I N TD => TD)
               (CHECK: (PAT -)(SNG -) )
               (SET: 1 (PAT +) )
               (SEMNOP)
               ) )
ADDRULE     ( (
               (I N TD => TD)
               (CHECK: (PAT -)(SNG +) )
               (SET: 1 (PAT +) )
               (SEMNOP)
               ) )
ADDRULE     ( (
               ( T H E AT => AT)
               (CHECK: (PAT -) )
               (SET: 1 )
               (SEMNOP)
               ) )
ADDRULE     ( (
               ( T H E TD => TD)
               (CHECK: (PAT -) )
               (SET: 1 )
               (SEMNOP)
```

```
                 ) )
ADDRULE ( (
                 ( 1 TD => TD)
                 (CHECK: (PAT -)(SNG +) )
                 (SET: 1 )
                 (SEMNOP)
                 ) )
ADDRULE ( (
                 ( A TD => TD)
                 (CHECK: (PAT -)(SNG +) )
                 (SET: 1 )
                 (SEMNOP)
                 ) )
ADDRULE ( (
                 (TD A N D TD => TD)
                 (CHECK: (DAT -)(MAT +)(PAT -) * (DAT -)(MAT +)(PAT -) )
                 (SET: 1 (SNG -) )
                 (SEMNOP)
                 ) )
ADDRULE ( (
                 ( U N T I L AT => AT)
                 (CHECK: (PAT -) )
                 (SET: 1 (PAT +) )
                 (SEMNOP)
                 ) )
ADDRULE ( (
                 ( F R O M AT => AT)
                 (CHECK: (PAT -) )
                 (SET: 1 (PAT +) )
                 (SEMNOP)
                 ) )
ADDRULE ( (
                 ( T C AT => AT)
                 (CHECK: (PAT -) )
                 (SET: 1 (PAT +) )
                 (SEMNOP)
                 ) )
ADDRULE ( (
                 ( T H R O U G H AT => AT)
                 (CHECK: (PAT -) )
                 (SET: 1 (PAT +) )
                 (SEMNOP)
                 ) )
ADDRULE ( (
                 ( F R O M TD => TD)
                 (CHECK: (DAT +)(MAT -)(PAT -) )
                 (SET: 1 (PAT +) )
                 (SEMNOP)
```

```
             ) )
ADDRULE  ( (
             (  T O TD => TD)
             (CHECK: (DAT +)(MAT -)(PAT -) )
             (SET: 1 (PAT +) )
             (SEMNOP)
             ) )
ADDRULE  ( (
             (  T H R O U G H TD => TD)
             (CHECK: (DAT +)(MAT -)(PAT -) )
             (SET: 1 (PAT +) )
             (SEMNOP)
             ) )
ADDRULE  ( (
             (  P R E V I O U S  T O AT => AT )
             (CHECK: (PAT -) )
             (SET: 1 (PAT +) )
             (SEMNOP)
             ) )
ADDRULE  ( (
             (  F O L L O W I N G AT => AT)
             (CHECK: (PAT -) )
             (SET: 1 (PAT +) )
             (SEMNOP)
             ) )
ADDRULE  ( (
             (  TD AT => AT)
             (CHECK: (SNG +)(PAT -) * (PAT +) )
             (SET: 2 (PAT -) )
             (SEMNOP)
             ) )
ADDRULE  ( (
             (  TD AT => AT)
             (CHECK: (SNG +)(PAT -) * (PAT -) )
             (SET: 2 )
             (SEMNOP)
             ) )
ADDRULE  ( (
             (  TD AT => AT)
             (CHECK: (SNG -)(PAT -) * (PAT +) )
             (SET: 2 (PAT -) )
             (SEMNOP)
             ) )
ADDRULE  ( (
             (  TD AT => AT)
             (CHECK: (SNG -)(PAT -) * (PAT -) )
             (SET: 2 )
             (SEMNOP)
```

```
       ) )
ADDRULE ( (
       ( TD AT => AT)
       (CHECK: (PAT +)(SNG +)(MAT -) * (PAT +)(PLU -) )
       (SET: 2 )
       (SEMNOP)
       ) )
ADDRULE ( (
       ( O F AT => AT)
       (CHECK: (PAT -) )
       (SET: 1 )
       (SEMNOP)
       ) )
ADDRULE ( (
       ( N E X T TD => AT)
       (CHECK: (PAT -)(SNG -) )
       (SET: 0 )
       (SEMNOP)
       ) )
ADDRULE ( (
       ( L A S T TD => AT)
       (CHECK: (PAT -)(SNG -) )
       (SET: 0 )
       (SEMNOP)
       ) )
ADDRULE ( (
       ( N O W => AT)
       (SET: 0 )
       (SEMNOP)
       ) )
ADDRULE ( (
       ( T O D A Y => AT)
       (SET: 0 )
       (SEMNOP)
       ) )
ADDRULE ( (
       (Y E S T E R D A Y => AT)
       (SET: 0 )
       (SEMNOP)
       ) )
ADDRULE ( (
       ( T O M O R R O W => AT)
       (SET: 0 )
       (SEMNOP)
       ) )
ADDRULE ( (
       ( B E F O R E VP => AT)
       (CHECK: (FSJ +)(FPP -) )
```

```
          (SET: 0 (PAT +) )
          (SEMNOP)
          ) )
ADCRULE ( (
          ( A F T E R VP => AT)
          (CHECK: (FSJ +)(FPP -) )
          (SET: 0 (PAT +) )
          (SEMNOP)
          ) )
ADCRULE ( (
          ( S I N C E VP => AT)
          (CHECK: (FSJ +)(FPP -) )
          (SET: 0 (PAT +) )
          (SEMNOP)
          ) )
ADCRULE ( (
          ( U N T I L VP => AT)
          (CHECK: (FSJ +)(FPP -) )
          (SET: 0 (PAT +) )
          (SEMNOP)
          ) )
ADCRULE ( (
          ( T H E  T I M E VP => AT)
          (CHECK: (FSJ +)(FPP -) )
          (SET: 0 (PAT +) )
          (SEMNOP)
          ) )
ADCRULE ( (
          ( T H E  T I M E  T H A T VP => AT)
          (CHECK: (FSJ +)(FPP -) )
          (SET: 0 (PAT +) )
          (SEMNOP)
          ) )
ADCRULE ( (
          ( T H E  L A S T  T I M E VP => AT)
          (CHECK: (FSJ +)(FPP -) )
          (SET: 0 (PAT +) )
          (SEMNOP)
          ) )
```

APPENDIX 4

SAMPLE RUN

```
  ARGUMENTS FOR EVALQUOTE ...

    PARSE ( (
     (DEF : H A R V A R D :=NAME) ))
COLLECTED     5885 CELLS

  TIME    2003 MS,  VALUE IS ....

    NIL

  ARGUMENTS FOR EVALQUOTE ...

    PARSE ( (
     (DEF : Y A L E :=NAME) ))
COLLECTED     5837 CELLS

  TIME    1443 MS,  VALUE IS ....

    NIL

  ARGUMENTS FOR EVALQUOTE ...

    PARSE ( (
     (DEF : B O Y :=NAME) ))
COLLECTED     5814 CELLS

  TIME    1264 MS,  VALUE IS ....

    NIL

  ARGUMENTS FOR EVALQUOTE ...

    PARSE ( (
     (DEF : S I S T E R :=RELATION) ))
COLLECTED     5826 CELLS
COLLECTED     5820 CELLS

  TIME    2365 MS,  VALUE IS ....

    NIL

  ARGUMENTS FOR EVALQUOTE ...
```

```
    PARSE ( (
     (DEF : A T T E N D :=VERB) ))
COLLECTED      5770 CELLS

  TIME    1940 MS,  VALUE IS ....

    NIL

  ARGUMENTS FOR EVALQUOTE ...

    PARSE ( (
     (I S   T H E R E   A   H A R V A R D   B O Y   W H O S E
         S I S T E R   A T T E N D S   Y A L E) ))
COLLECTED      5659 CELLS
COLLECTED      5625 CELLS
COLLECTED      5645 CELLS
COLLECTED      5521 CELLS
COLLECTED      5495 CELLS
COLLECTED      5615 CELLS
COLLECTED      5454 CELLS
COLLECTED      5305 CELLS
COLLECTED      5470 CELLS
COLLECTED      5451 CELLS
COLLECTED      5416 CELLS
COLLECTED      5424 CELLS
COLLECTED      5496 CELLS
COLLECTED      5218 CELLS
COLLECTED      5222 CELLS
COLLECTED      5135 CELLS

  TIME   19193 MS,  VALUE IS ....

    ((VP NAME (IS THERE (A ((HARVARD BOY) WHOSE SISTER (ATTENDS YALE))))
    FVI + FQT + FPH + FSJ + FSI +))

  ARGUMENTS FOR EVALQUOTE ...

*** END OF DATA ***
```

LIST OF REFERENCES

1. Chomsky, Noam, Aspects of the Theory of Syntax. MIT Press, Cambridge, Massachusetts, 1965, 251 p.

2. Bolce, J. F., "LISP/360: A Description of the University of Waterloo LISP 1.5 Interpreter for the IBM System/360", University of Waterloo, Waterloo, Ontario, October 1967.

3. Dostert, B. H., "REL — An Information System for a Dynamic Environment", REL Report No. 3, California Institute of Technology, Pasadena, California, December 1971.

4. Dostert, B. H., and Thompson, F. B., "Syntactic Analysis in REL English", Proceedings, 1971 International Meeting on Computational Linguistics, Debrecen, Hungary, September 1971.

5. Dostert, B. H., and Thompson, F. B., "The Syntax of REL English", REL Report No. 1, California Institute of Technology, Pasadena, California, September 1971.

6. Harman, G. H., "Generative Grammars without Transformational Rules: A Defense of Phrase Structure.", Language, 39, p. 597-616.

7. Merriam, E. W., "Users Guide for the General Purpose Parser (Version 6.5)", Pennsylvania State University, University Park, Pennsylvania.

8. Minsky, Marvin, Semantic Information Processing, MIT Press, Cambridge, Massachusetts, 1968, 440 p.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Asst Professor G. D. Gibbons, Code 53 gp Department of Mathematics Naval Postgraduate School Monterey, California 93940	2
4.	Lt. David P. Courts, USN 127 Povelton Avenue Woodlynne, New Jersey 08107	1

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

IGINATING ACTIVITY *(Corporate author)* Naval Postgraduate School Monterey, California 93940	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

PORT TITLE

Natural Language System for Experimenting with Semantic Representa-
ons based on the REL English Syntax and a General Purpose Parser
LISP

SCRIPTIVE NOTES *(Type of report and, inclusive dates)*

ster's Thesis; March 1973

THOR(S) *(First name, middle initial, last name)*

vid Paul Courts

PORT DATE rch 1973	7a. TOTAL NO. OF PAGES 82	7b. NO. OF REFS 8

ONTRACT OR GRANT NO. ROJECT NO.	9a. ORIGINATOR'S REPORT NUMBER(S) 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

ISTRIBUTION STATEMENT

proved for public release; distribution unlimited.

JPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Naval Postgraduate School Monterey, California 93940

BSTRACT

A system for experimenting with semantic representation is
eveloped. The existing REL System is discussed and comparisons are
de. The proposed system, written in LISP, is described. A method
r extending the user's language is developed. Recommendations for
rther study are presented. Program listings and sample results
e included in the appendices.

FORM 1 NOV 65 1473 (PAGE 1)

N)101-807-6811

Security Classification

A-31408

Security Classification

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Natural Language Processing						
Semantic Representations						
REL System						
General Purpose Parser						
LISP						

Thesis 141959
C75675 Courts
c.1 A natural language system for experimenting with semantic representations based on the REL English syntax and a general purpose parser in LISP.

4 AUG 76 24228
18 OCT 84 29815
23 JAN 86 33197

141959
Thesis
C75675 Courts
c.1 A natural language system for experimenting with semantic representations based on the REL English syntax and a general purpose parser in LISP.

thesC75675
A natural language system for experiment
3 2768 002 08990 6
DUDLEY KNOX LIBRARY